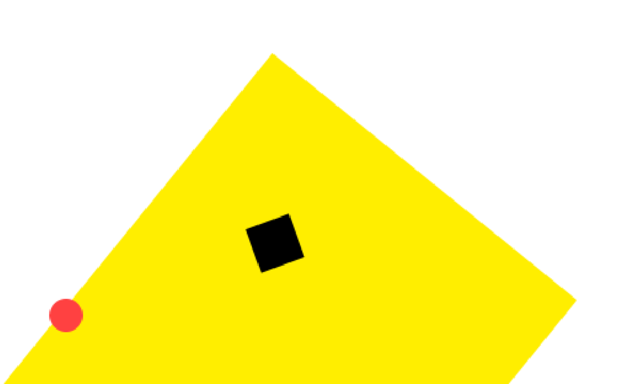
Пишем компоненты, которые захочется переиспользовать

Антон Крылов

Авито





Кто я

Frontend developer



Кто я

Frontend developer

Люблю архитектурить и развлекаться за ноутбуком по ночам



Кто я

Frontend developer

Люблю архитектурить и развлекаться за ноутбуком по ночам

- Бывший тимлид



Кто я

Frontend developer

Люблю архитектурить и развлекаться за ноутбуком по ночам

- Бывший тимлид
- Делаю профили в Avito



Идеальный компонент



Идеальный компонент

- Который не стали писать



Идеальный компонент

- Который не стали писать
- На который есть дока и тесты



Идеальный компонент

- Который не стали писать
- На который есть дока и тесты
- При использовании не нужно думать



Идеальный компонент

- Который не стали писать
- На который есть дока и тесты
- При использовании не нужно думать
- Подходит под все сценарии использования




Нельзя просто взять и написать


Сложность и неконсистентность
avito.tech
12

График работы программиста



Нет тестов



Нет тестов

- Скорее всего разработчик не подумал о том, как его компонент будет использоваться извне



Нет тестов

- Скорее всего разработчик не подумал о том, как его компонент будет использоваться извне
- Даже если подумал, возможно, он не подумал как потом этот компонент расширять

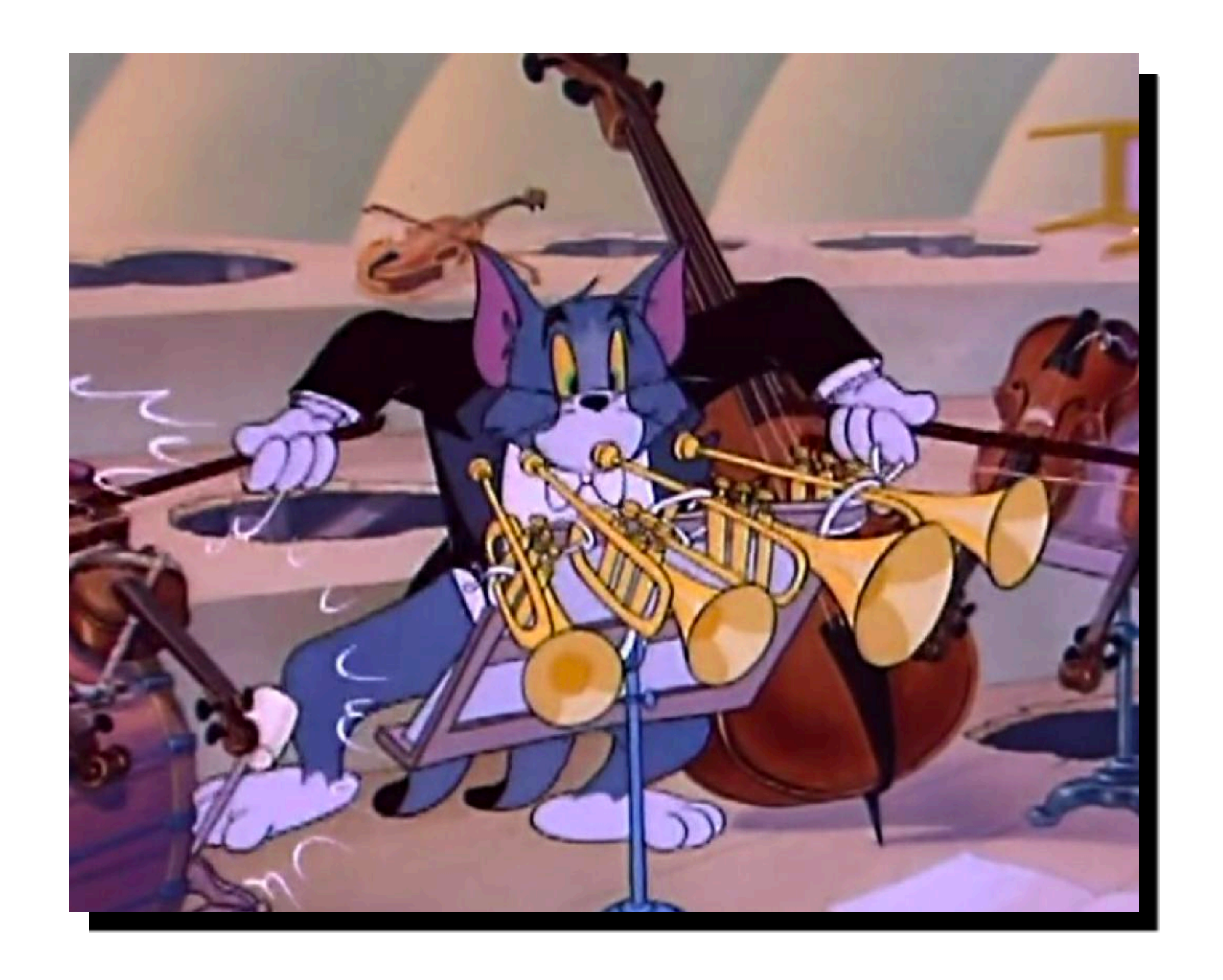


Компонент делает слишком много



Компонент делает слишком много

- Декомпозируйте, пожалуйста



Over engineering



Over engineering

- useEffect вместо useCallback



Over engineering

- useEffect вместо useCallback
- RxJS для всего



Over engineering

- useEffect вместо useCallback
- RxJS для всего
- Лишние слои абстракции

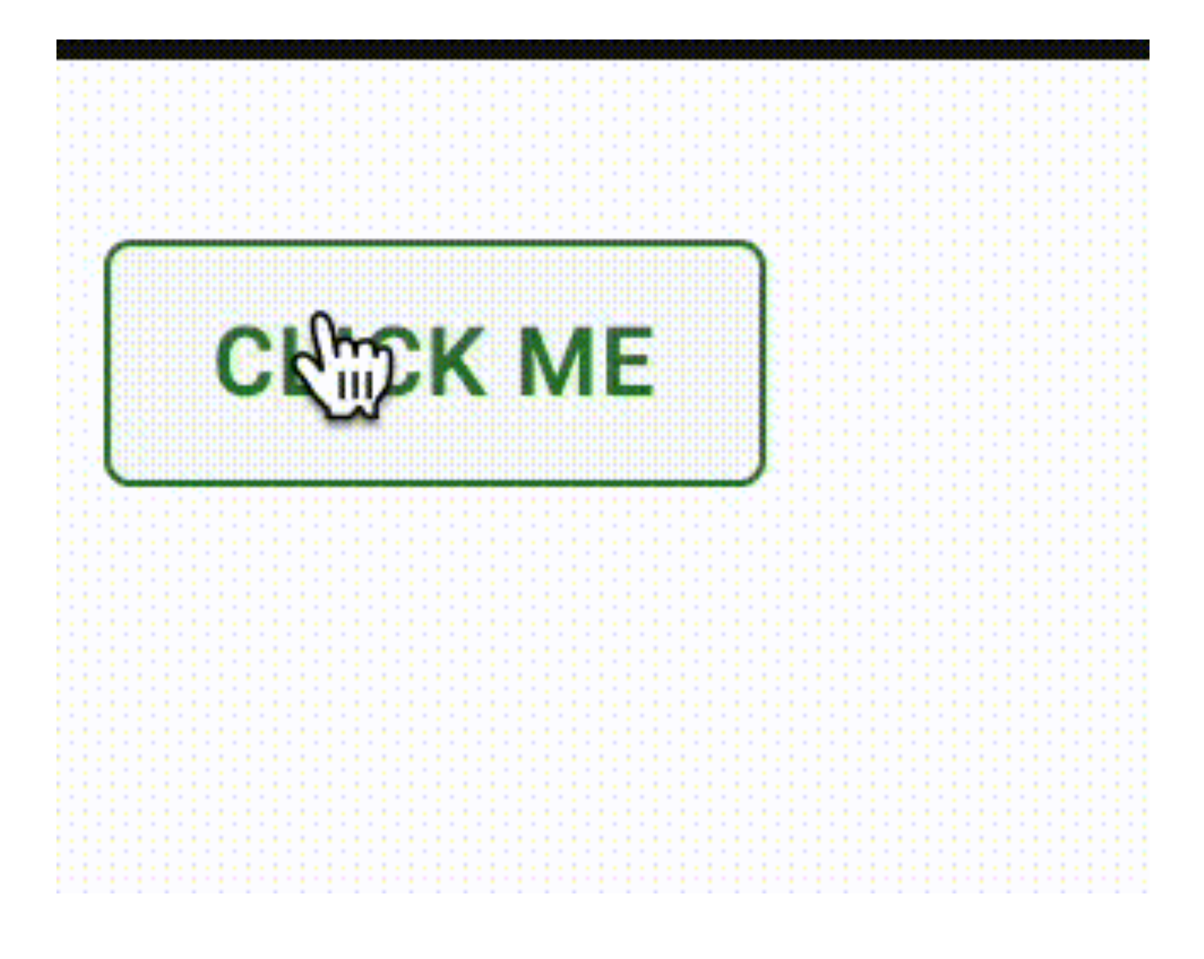


Over engineering

- useEffect вместо useCallback
- RxJS для всего
- Лишние слои абстракции
- Пока вы всё не прочитаете, не поймете, как оно работает



Неконсистентное состояние



Неконсистентное состояние

Что не так?)

```
export interface TextFieldProps extends
Omit<React.HTMLProps<HTMLInputElement>,
'onChange'> {
    ...
    errorText?: string;
    validation?: (value: string) => boolean;
    ...
}
```

Неконсистентное состояние

А тут?

```
export interface TextFieldProps extends
Omit<React.HTMLProps<HTMLInputElement>,
'onChange'> {
    ...
    validationProps?: {
        errorText: string;
        validation: (value: string) => boolean;
    },
    ...
}
```

Неконсистентное состояние

Решение: **single prop - single logic**

```
export interface TextFieldProps extends
Omit<React.HTMLProps<HTMLInputElement>,
'onChange'> {
    ...
    validation?: (value: string) => string;
    ...
}
```

Отображение на флагах

```
return isCreateForm ? <CreateForm /> : <EditForm />
```

Отображение на флагах

Если уж делаете, то сделайте хотя бы маппинг :)

```
const forms = {
    createForm: CreateForm,
    editForm: EditForm
};

interface Props {
    formName: keyof typeof forms;
}

const Form = ({ formName }: Props) => {
    const Component = forms[formName];

    return <Component />
}
```

Props hell

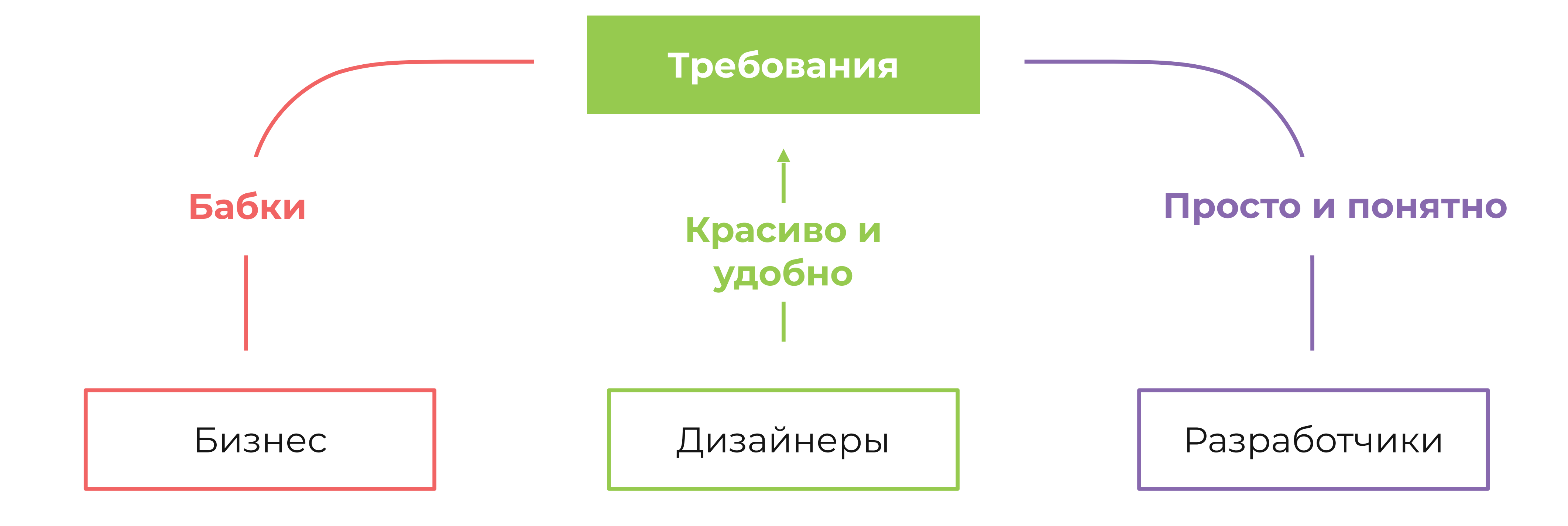
Сходу не разберешься, что происходит)

```
function TextField({
    withoutImplicitFocus,
    disabled,
    onFocus,
    hasLowerCase,
    hasAutoSelectAfterSubmit,
    onChange: onChangeProp,
    hasAutoSelect = true,
    selector = DEFAULT_SELECTOR,
    inputSize = "l",
    priority = 0,
    dataE2e = selector || DEFAULT_SELECTOR,
    dataTestId = selector || DEFAULT_SELECTOR,
    handleEnter = selectOnEnter,
    transformValueOnChange = transformToUppercase,
    onKeyDown = noop,
    useSuperFocus = useSuperFocusDefault,
    useFocusAfterError = useFocusAfterErrorDefault,
    useSuperFocusOnKeydown = useSuperFocusOnKeydownDefault,
    useSuperFocusAfterDisabled = useSuperFocusAfterDisabledDefault,
    someJSX,
    ...textFieldProps
}: Props)
```

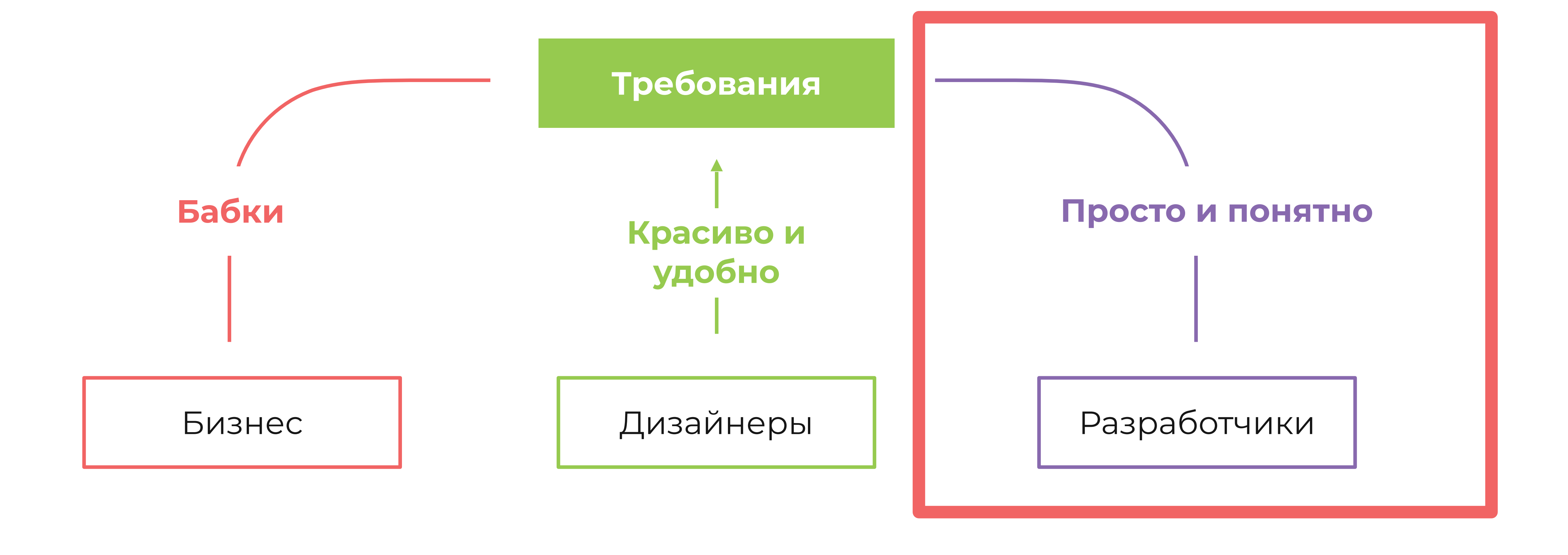

ДЕЛАЙТЕ ХОРОШО
А ПЛОХО НЕ ДЕЛАЙТЕ


Зачем
avito.tech

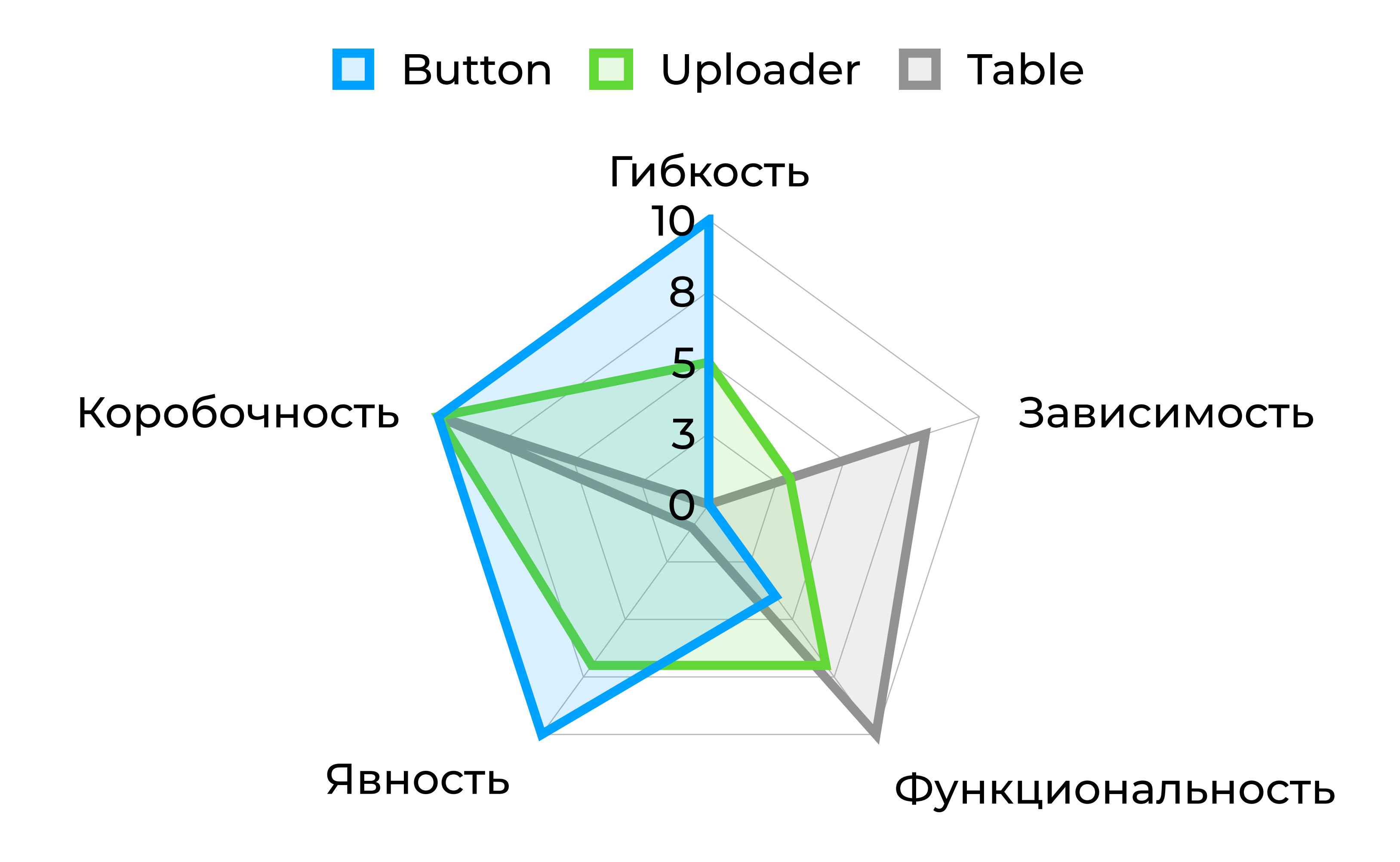
Требования


Требования
Бабки
Красиво и удобно
Просто и понятно
Бизнес
Дизайнеры
Разработчики

Требования


Требования
Бабки
Красиво и удобно
Просто и понятно
Бизнес
Дизайнеры
Разработчики

Характеристики



Гибкость

Какую часть можно переопределить

```
function TextField({
    withoutImplicitFocus,
    disabled,
    onFocus,
    hasLowerCase,
    hasAutoSelectAfterSubmit,
    onChange: onChangeProp,
    hasAutoSelect = true,
    selector = DEFAULT_SELECTOR,
    inputSize = "l",
    priority = 0,
    dataE2e = selector || DEFAULT_SELECTOR,
    dataTestId = selector || DEFAULT_SELECTOR,
    handleEnter = selectOnEnter,
    transformValueOnChange = transformToUppercase,
    onKeyDown = noop,
    useSuperFocus = useSuperFocusDefault,
    useFocusAfterError = useFocusAfterErrorDefault,
    useSuperFocusOnKeydown = useSuperFocusOnKeydownDefault,
    useSuperFocusAfterDisabled = useSuperFocusAfterDisabledDefault,
    someJSX,
    ...textFieldProps
}: Props)
```

Коробочность

Простота внешнего API

```
interface Props {
    entryUrl: string;
}

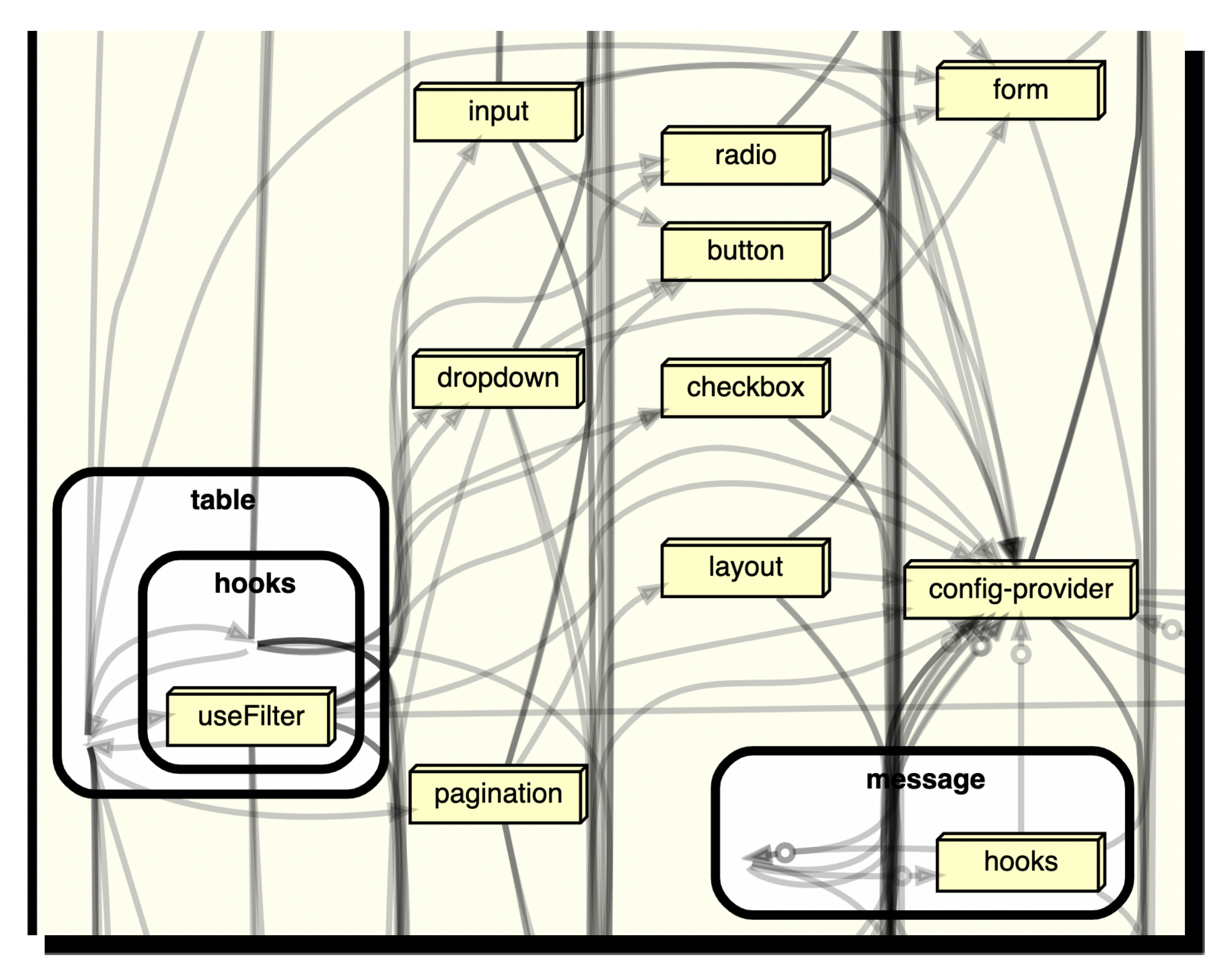
const Table = ({ entryUrl }: Props) => {
    const { rows, headers } = useDataFromUrl(entryUrl);
    ...
}
```

Коробочность

Простота внешнего API

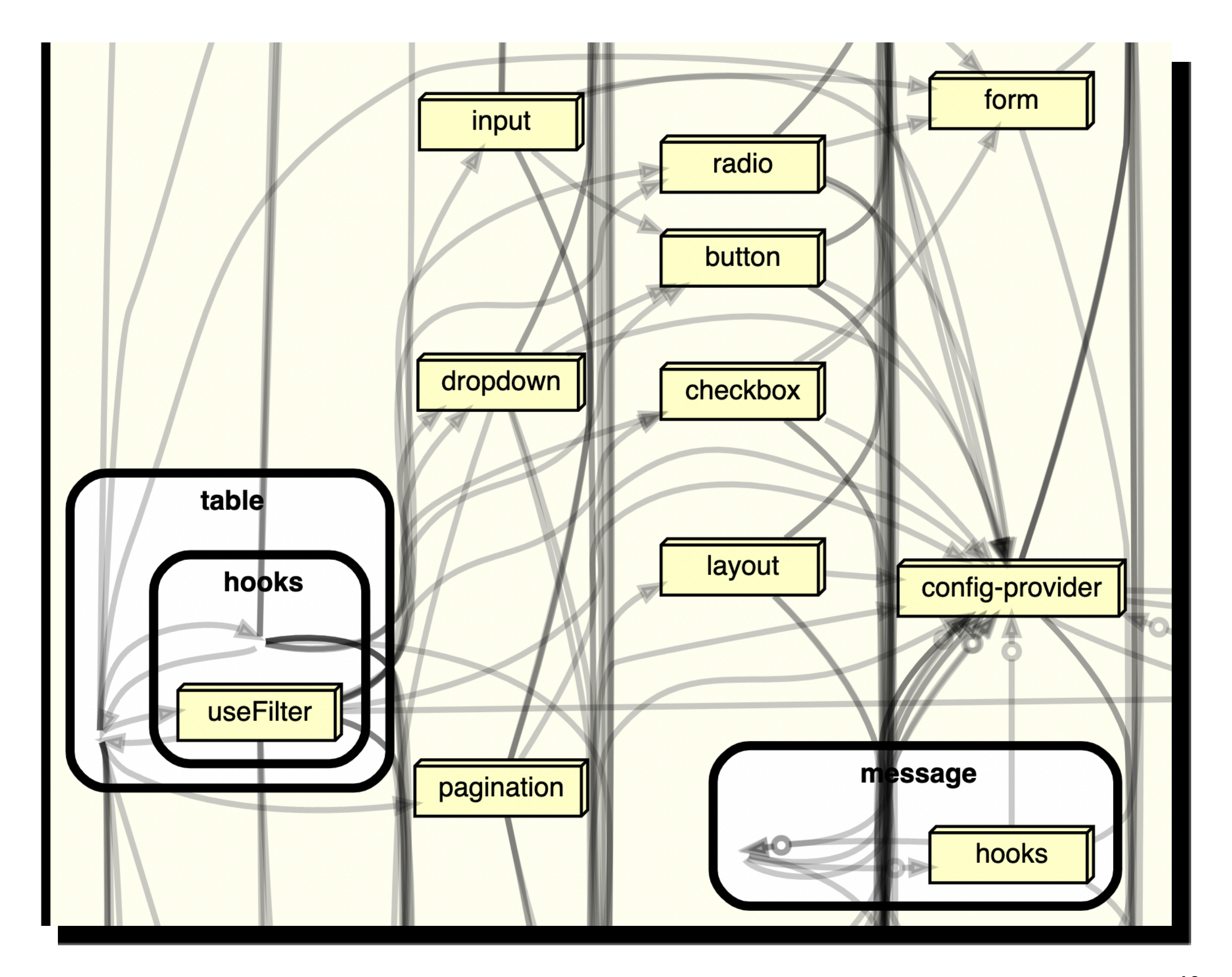
```
const PrimaryButton = ({ children }: Props) => {
    return (
        <Button variant='primary'>{children}</Button>
    )
}
```

Зависимость



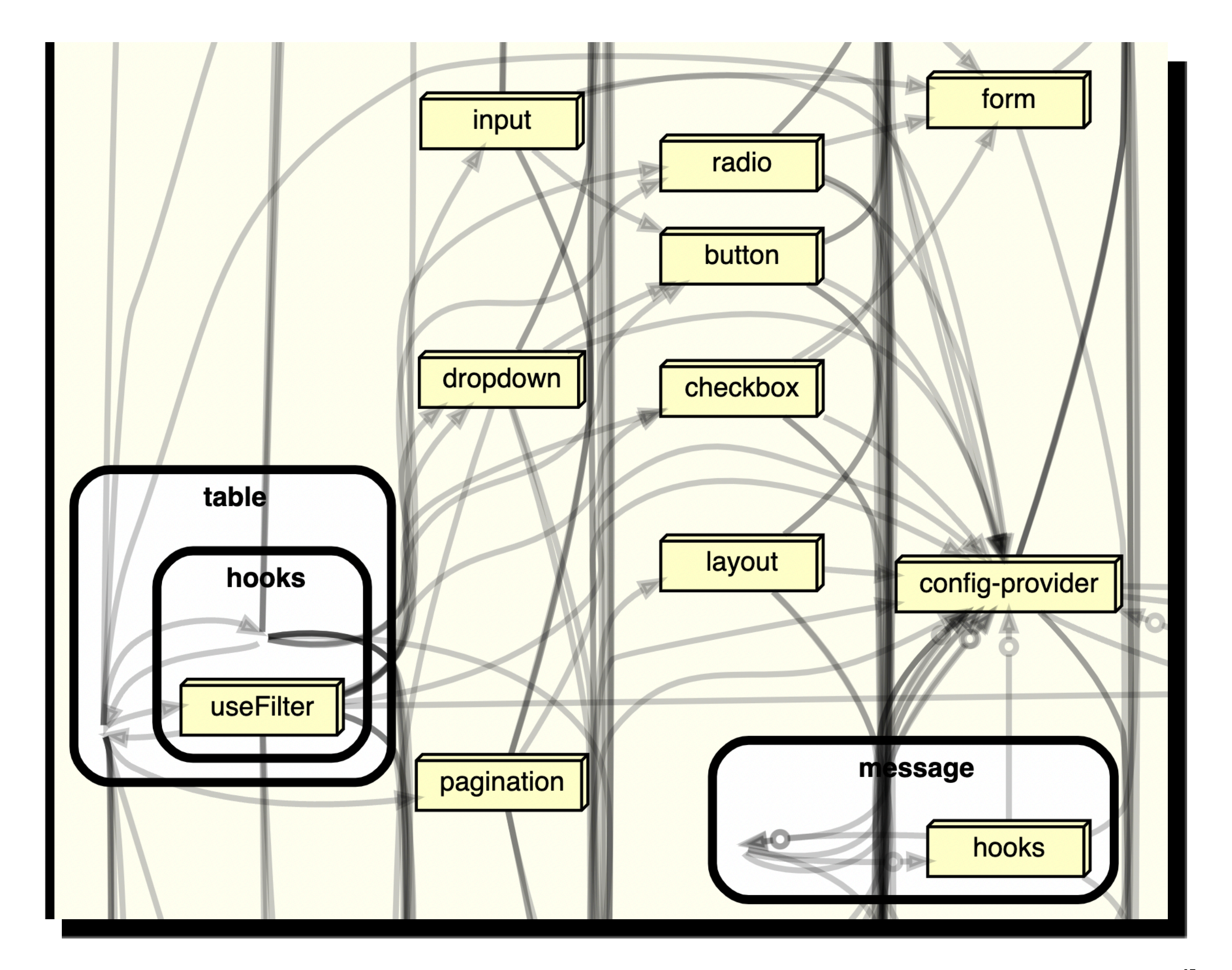
Зависимость

- Менее гибко



Зависимость

- Менее гибко
- Более хрупко :)



Явность

Необходимы ли знания контекста или бизнес логики?

```
const PrimaryButton = ({ children }: Props) => {
    return (
        <Button variant='primary'>{children}</Button>
    )
}
```

Явность

Необходимы ли знания контекста или бизнес логики?

```
const PrimaryButton = ({ children }: Props) => {
  return (
    <Button variant='primary'>{children}</Button>
  )
}

const PrimaryButton = ({ children }: Props) => {
  return (
    <Button color='red50'>{children}</Button>
  )
}
```

Явность

Необходимы ли знания контекста или бизнес логики?

```
const PrimaryButton = ({ children }: Props) => {
  return (
    <Button variant='primary'>{children}</Button>
  )
}

const PrimaryButton = ({ children }: Props) => {
  return (
    <Button color='red50'>{children}</Button>
  )
}

const PrimaryButton = ({ children }: Props) => {
  return (
    <Button color='#f00'>{children}</Button>
  )
}
```

Явность

Необходимы ли знания контекста или бизнес логики?

```
const Avatar = ({ isShop, src }: Props) => {
  return (
    <Image
      aspectRatio={isShop ? '3 / 2' : '1 / 1'}
      src={src}
    />
  )
}

const Avatar = ({ shape, src }: Props) => {
  return (
    <Image
      aspectRatio={shape === 'rectangle' ? '3 / 2' : '1 / 1'}
      src={src}
    />
  )
}
```

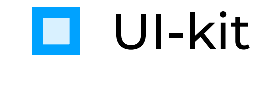
Features

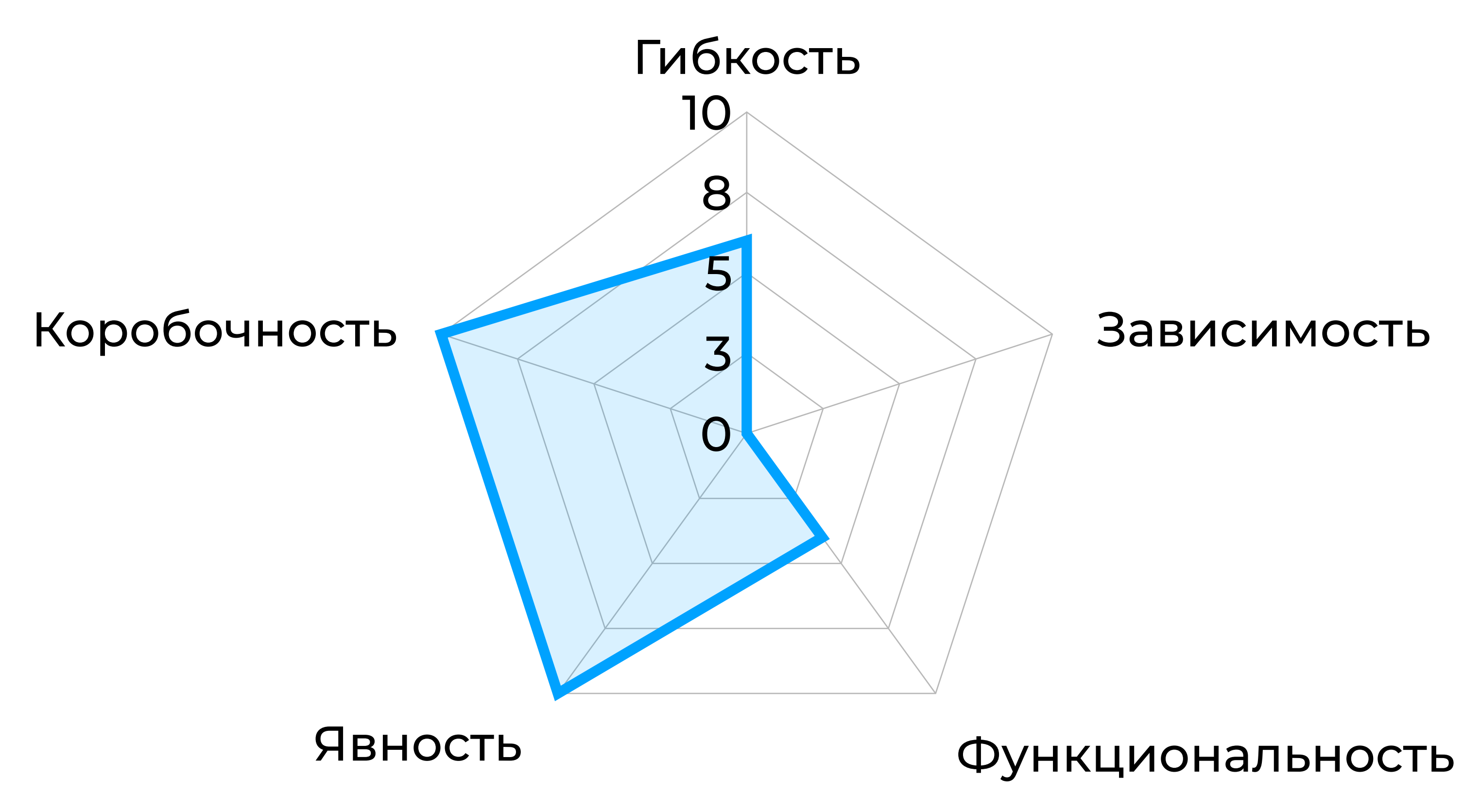
Количество возможностей, которое предоставляет нам компонент

```
function BasicForm() {
    ...
    const form = useForm({
      onSubmit(values) {
        alert(JSON.stringify(values, null, 2))
        console.log('values', values)
      },
      children: [
        {
          label: 'First Name',
          name: 'firstName',
          component: 'Input',
          value: '',
        },
        {
          component: 'Submit',
          text: 'submit',
        },
      ],
    })

    return <Form form={form} />
}
```

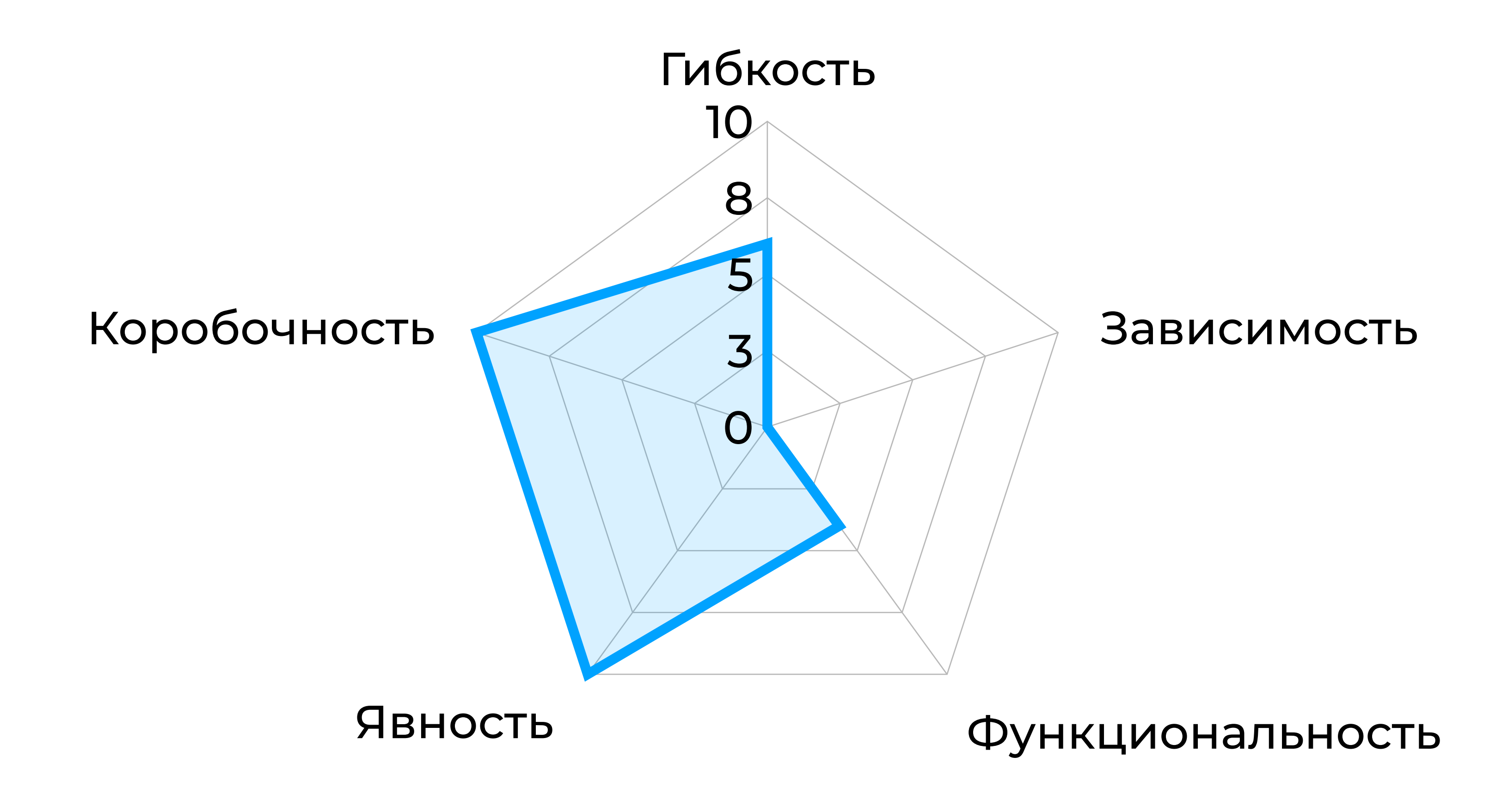
UI kit


UI-kit


Гибкость
10
8
5
3
0
Зависимость
Функциональность
Явность
Коробочность

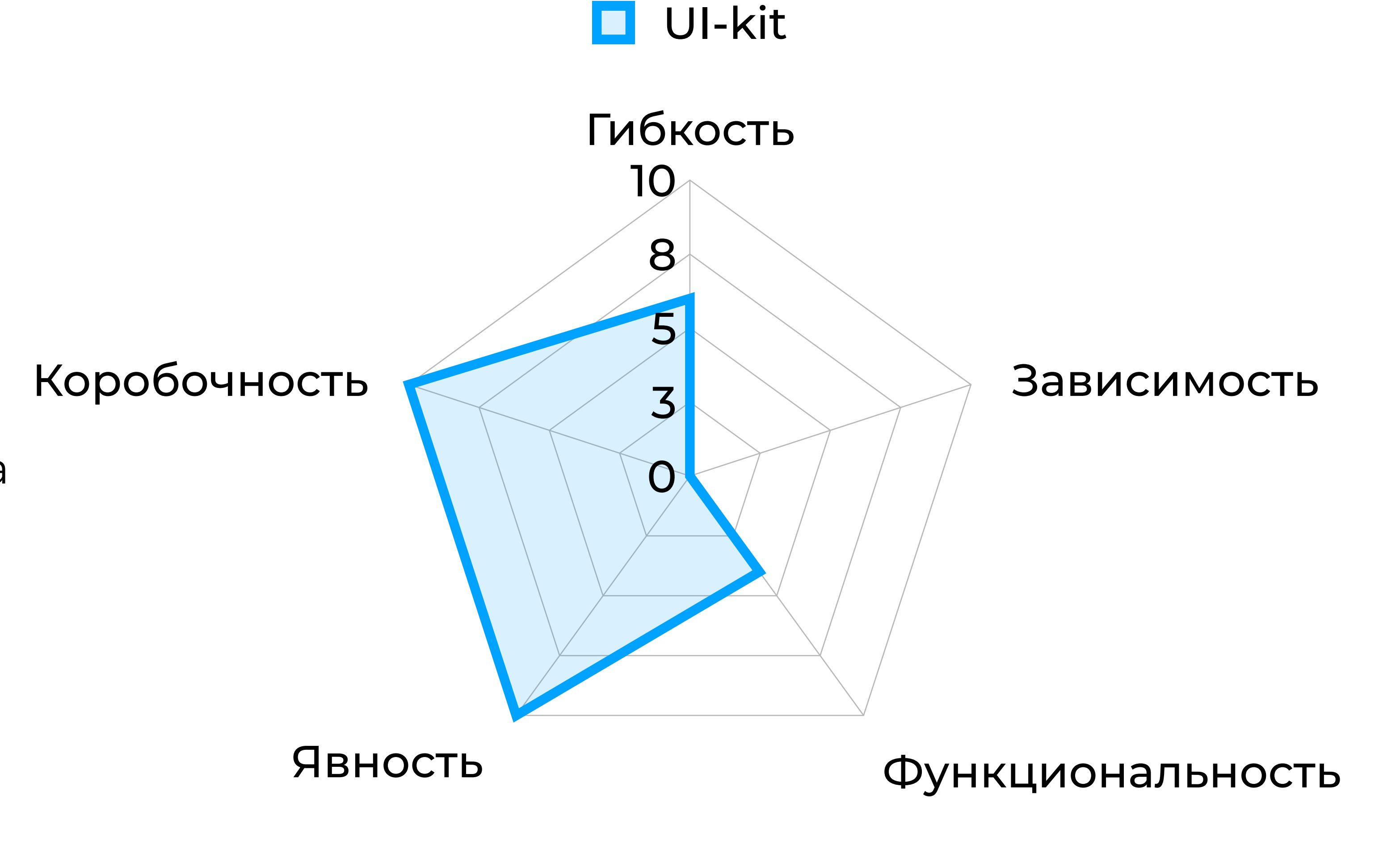
UI kit

- Ограничиваем возможность переопределения стилей на уровне темы



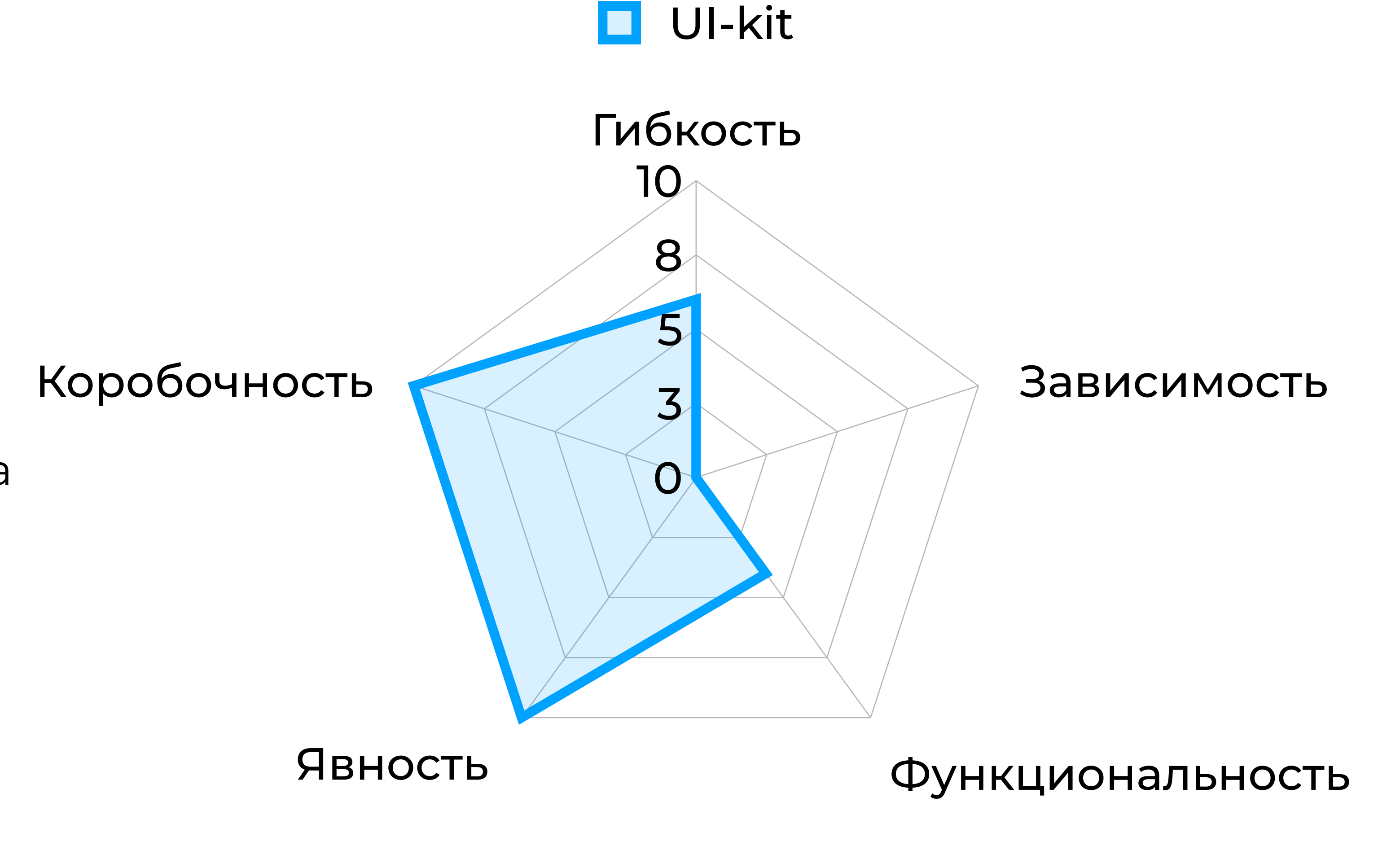
UI kit

- Ограничиваем возможность переопределения стилей на уровне темы
- Zero-dependency, чтобы весила поменьше

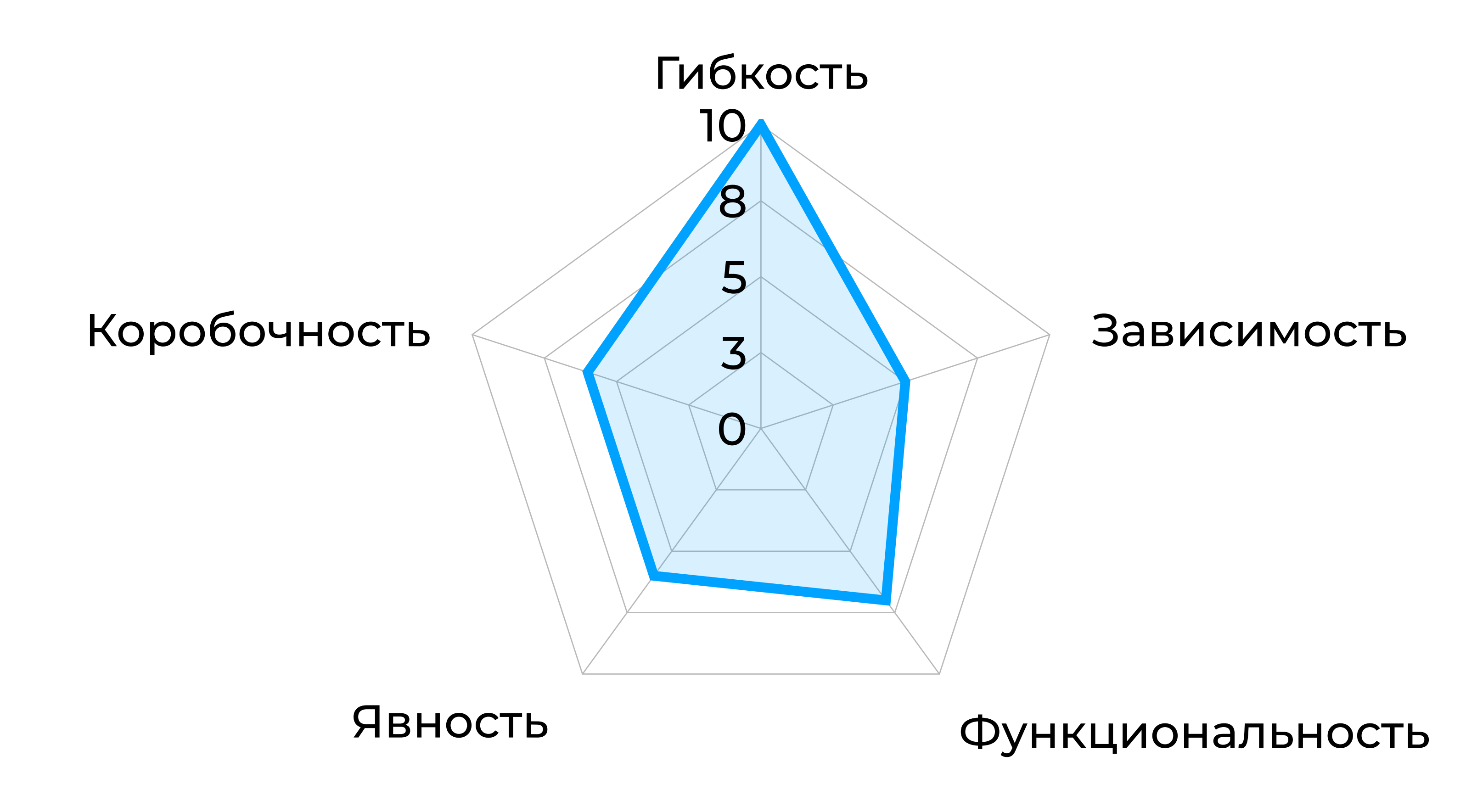


UI kit

- Ограничиваем возможность переопределения стилей на уровне темы
- Zero-dependency, чтобы весила поменьше
- Должно быть очень простое и явное API

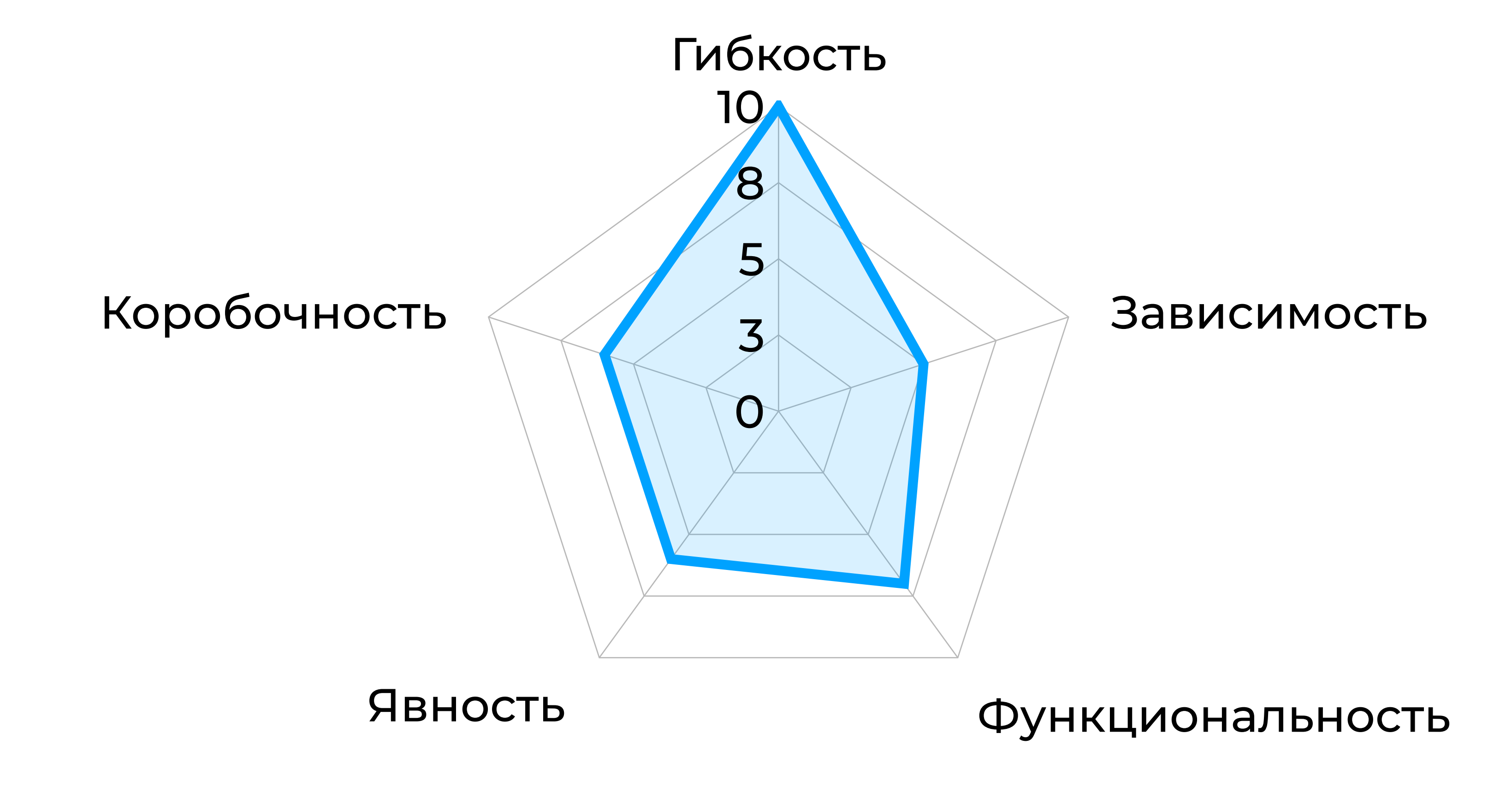


Небольшое количество проектов



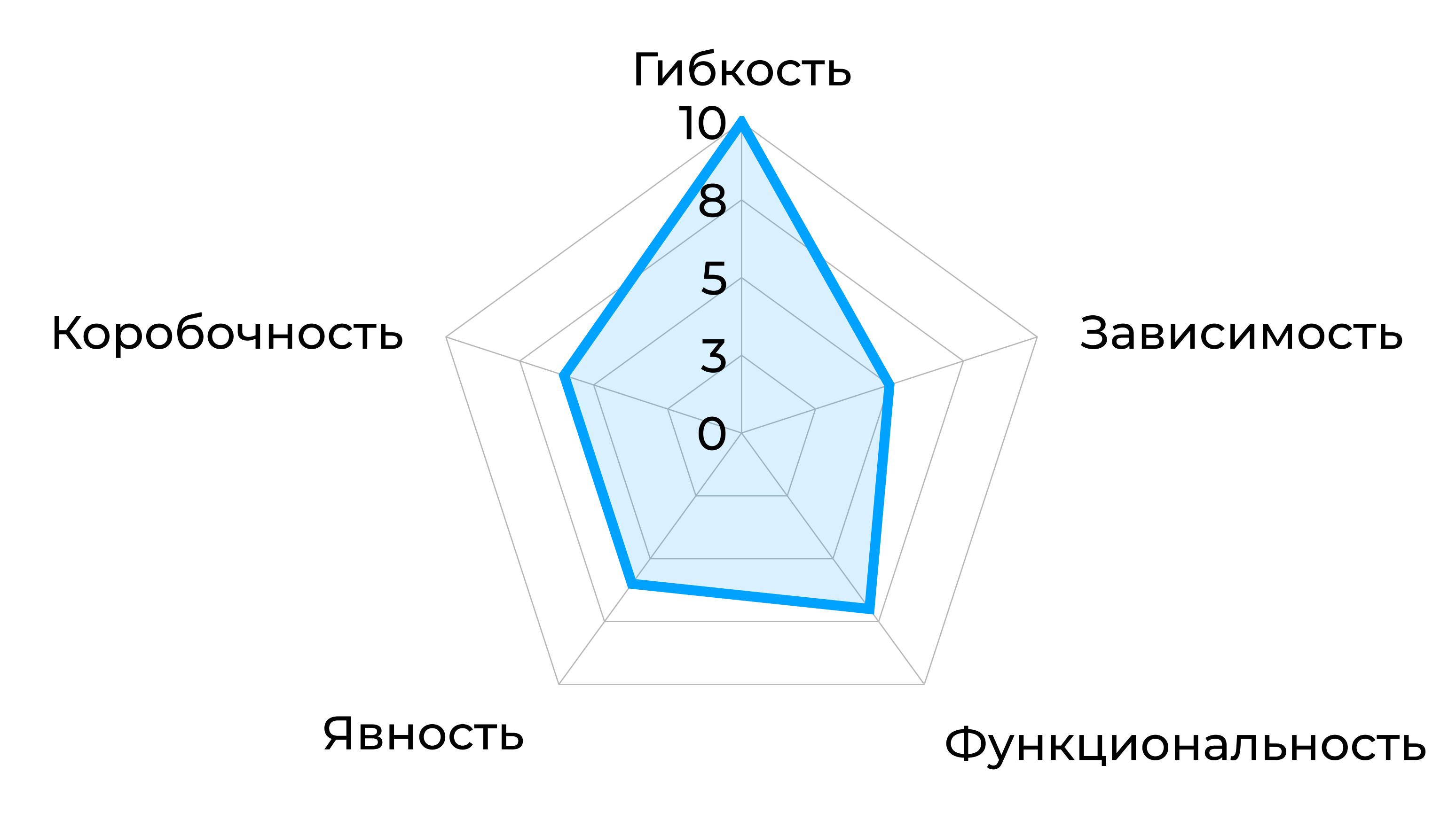
Небольшое количество проектов

- Максимальный уровень переопределения под бизнес логику



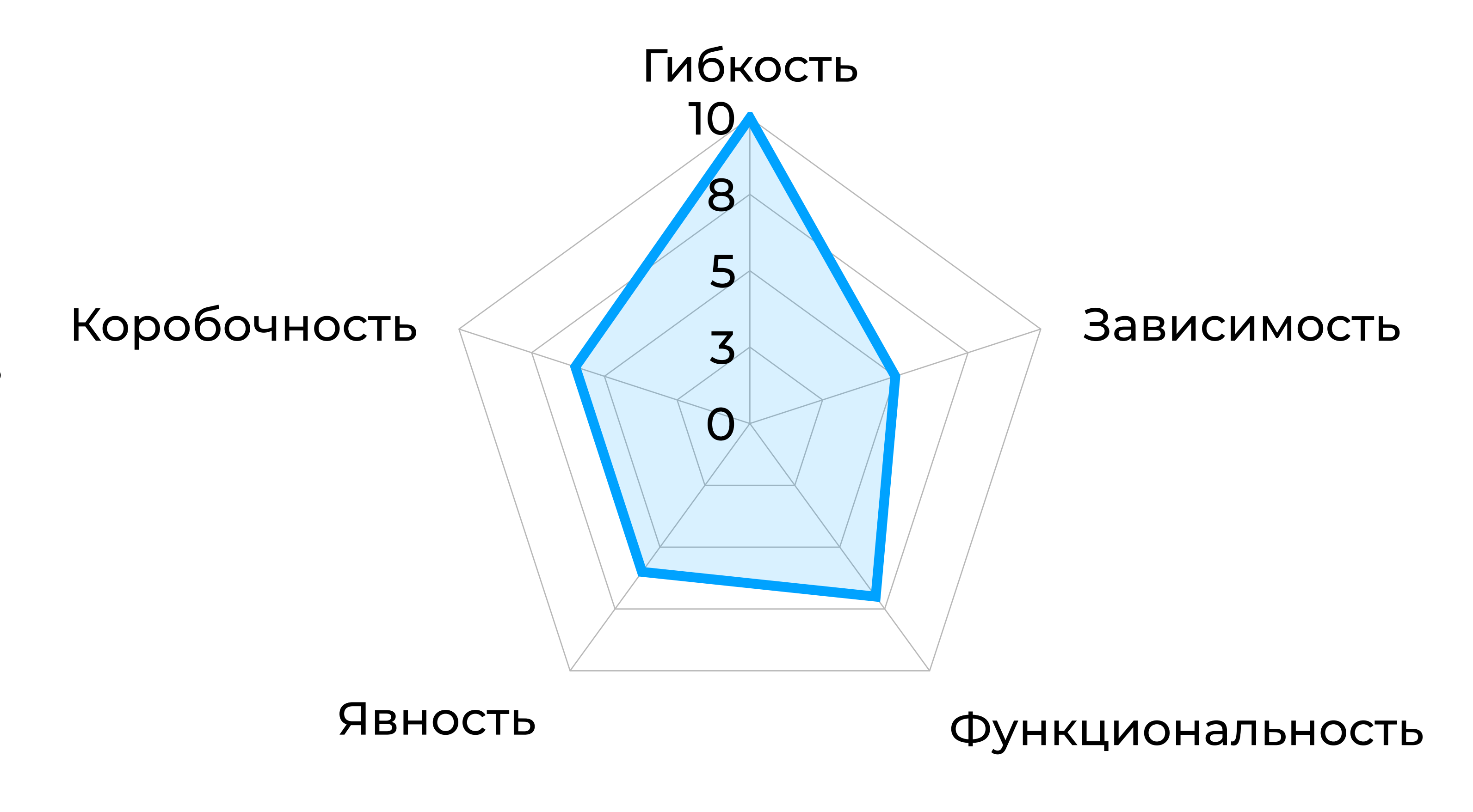
Небольшое количество проектов

- Максимальный уровень переопределения под бизнес логику
- Функциональности побольше, чтобы делать меньше на уровне проектов

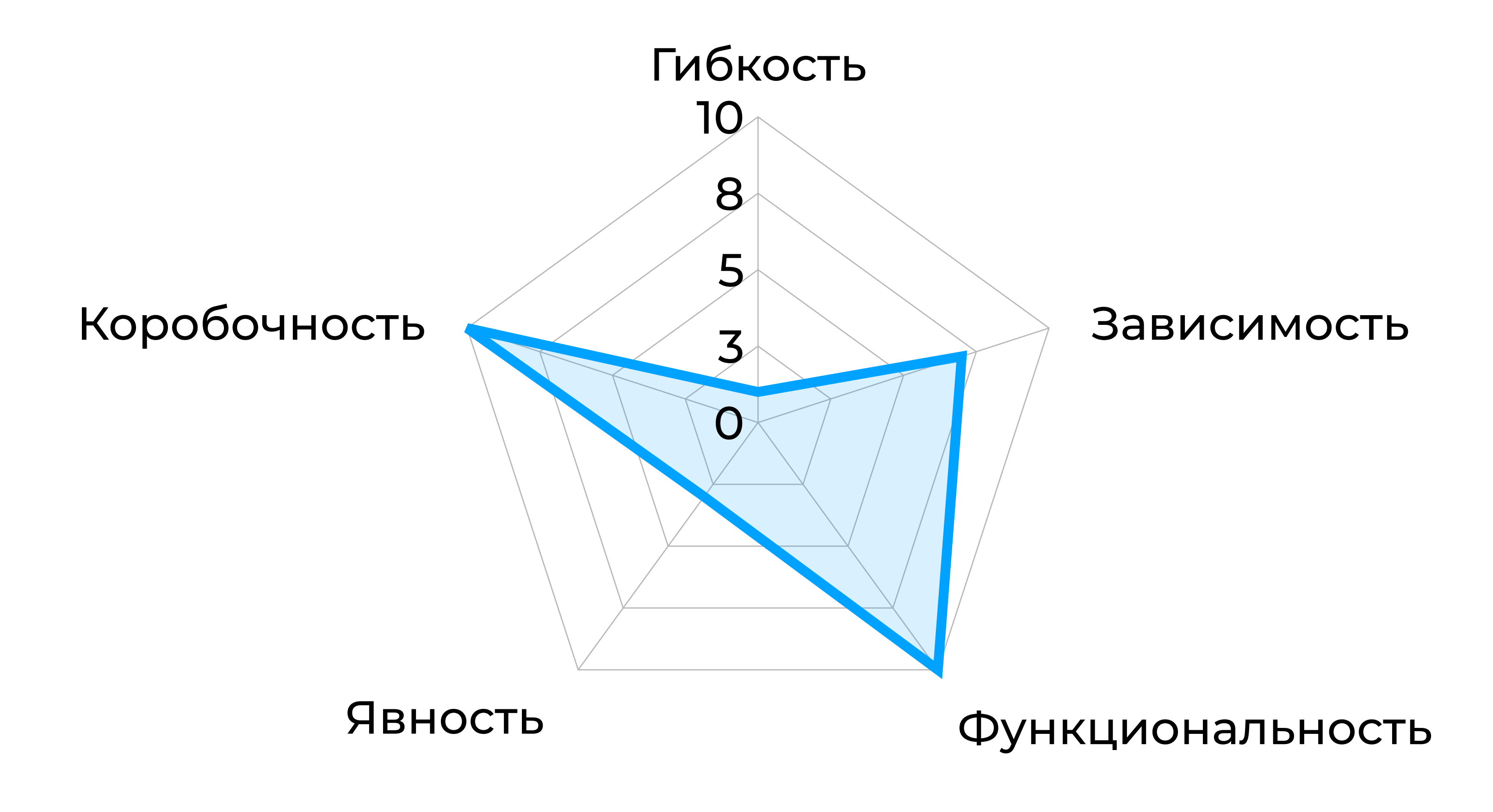


Небольшое количество проектов

- Максимальный уровень переопределения под бизнес логику
- Функциональности побольше, чтобы делать меньше на уровне проектов
- API может быть не очень явным, в пользу большей функциональности

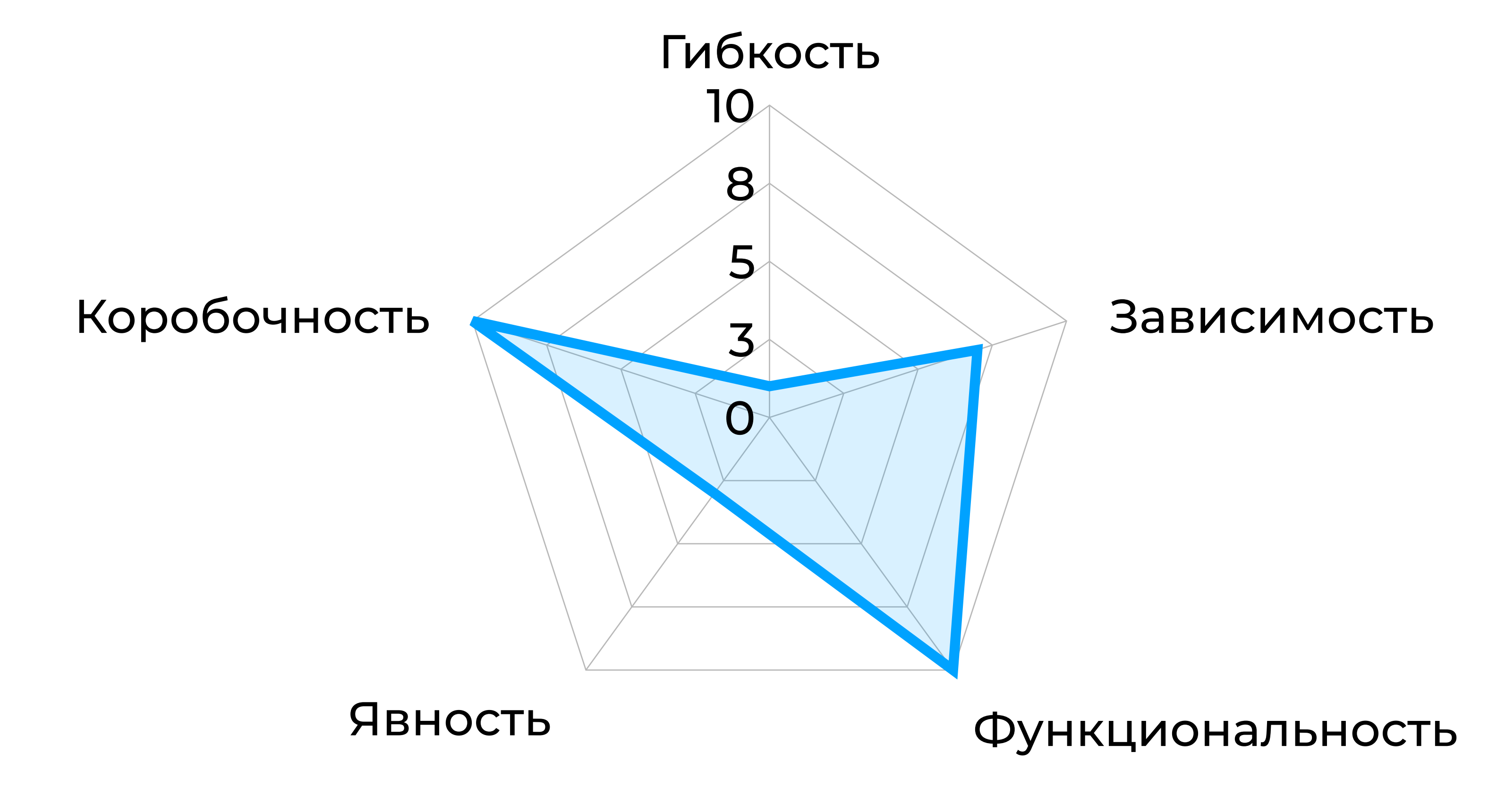


Локальные компоненты



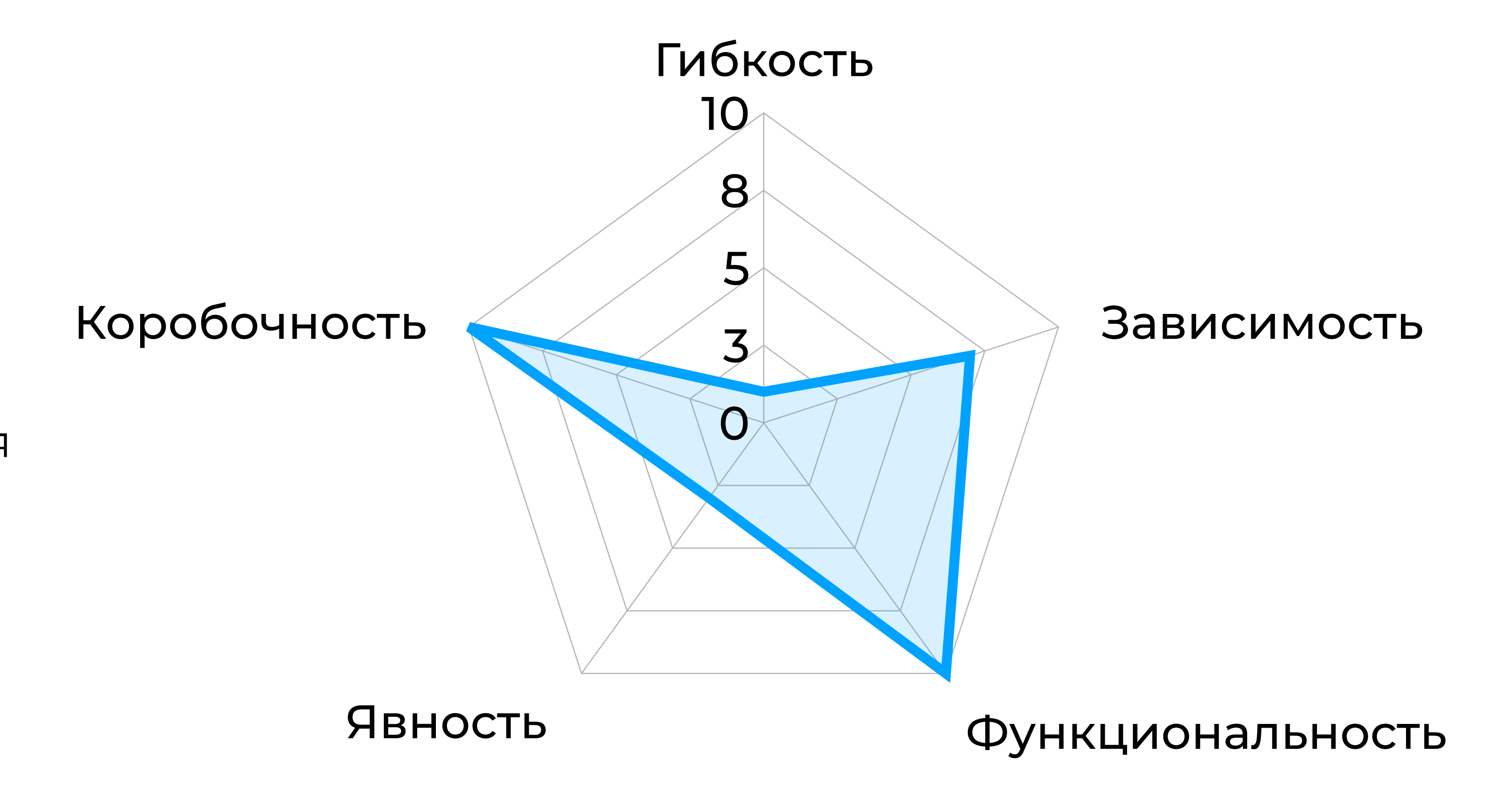
Локальные компоненты

- Могут быть максимально не гибкими, ведь мы можем их в любой момент поправить



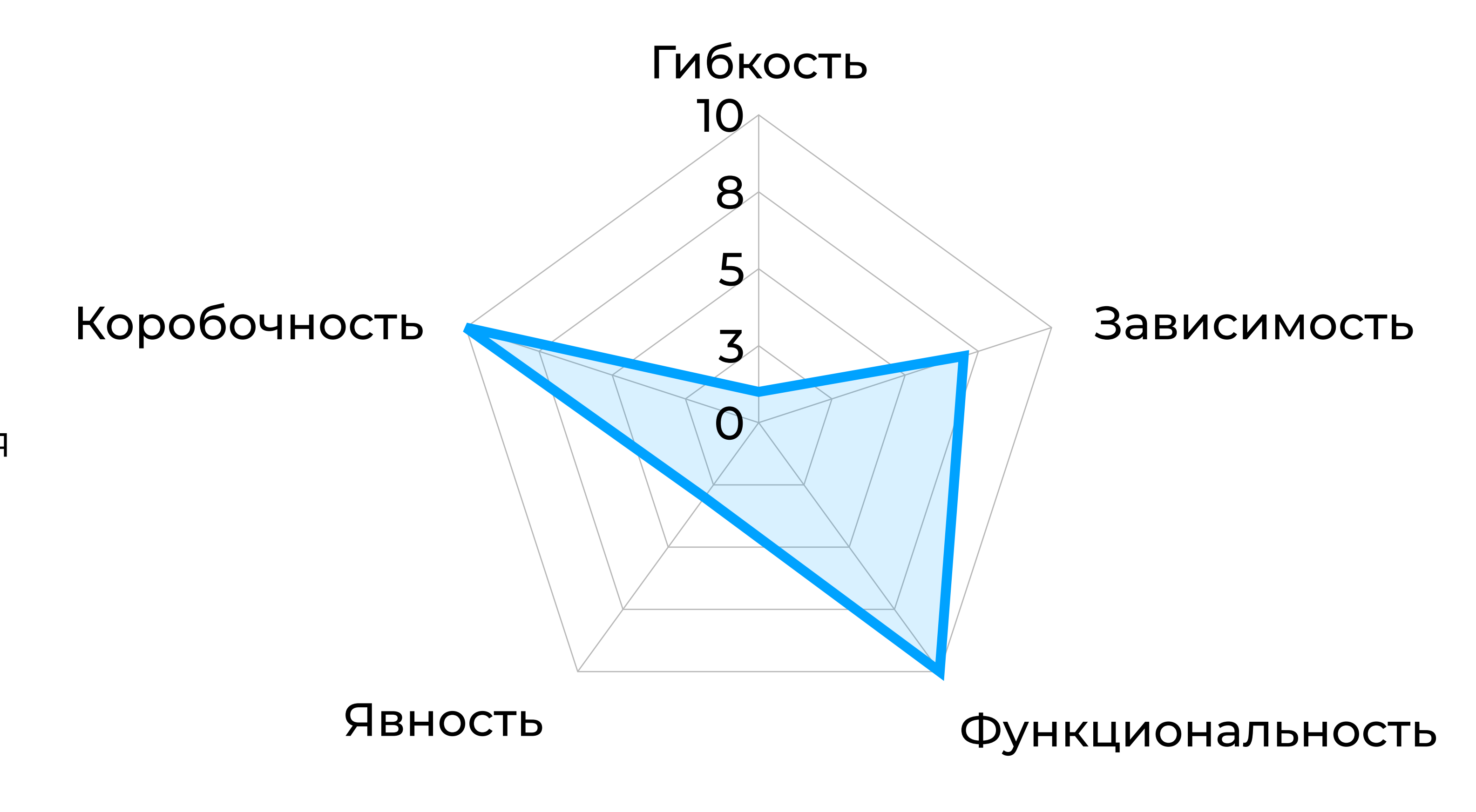
Локальные компоненты

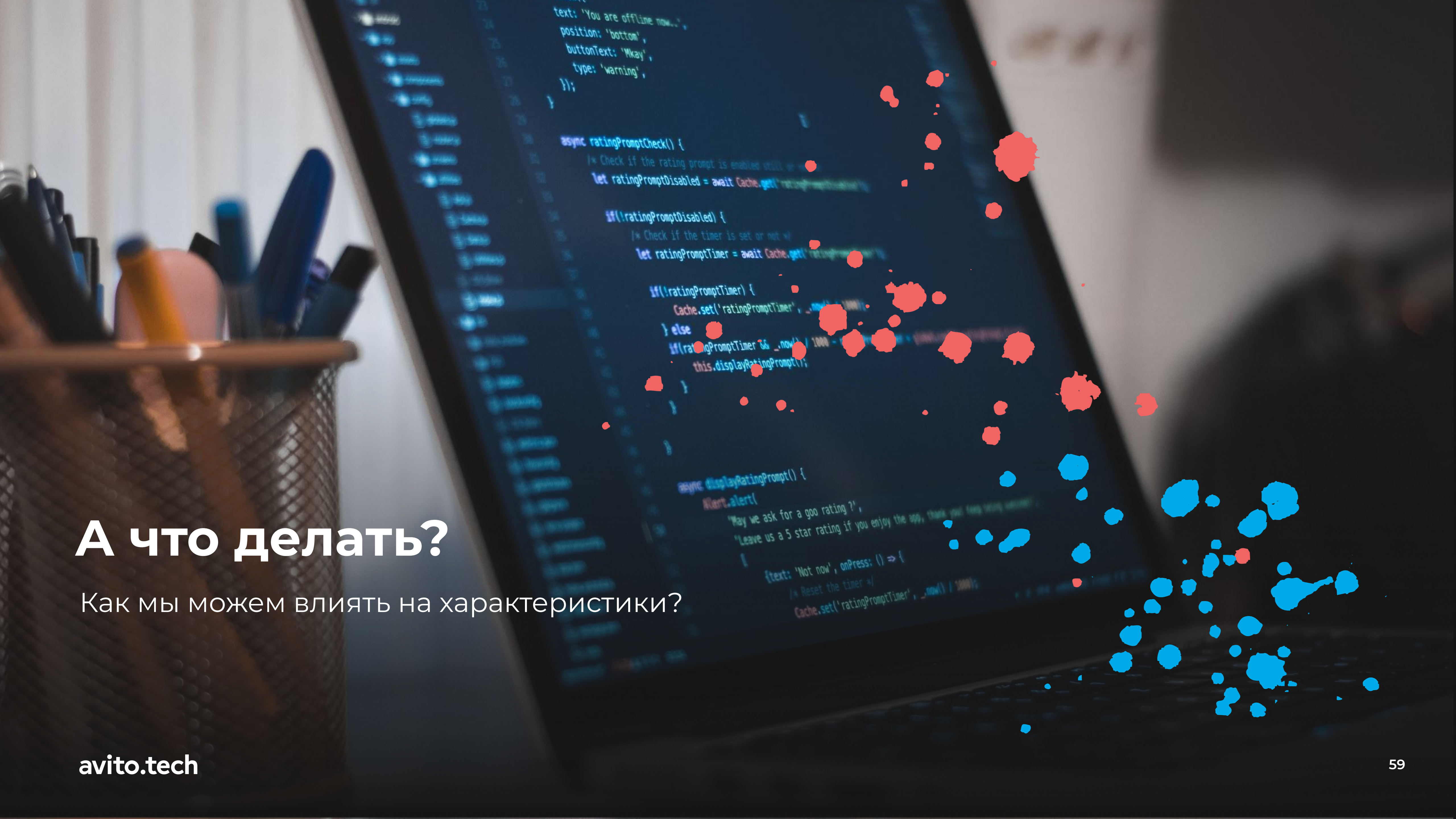
- Могут быть максимально не гибкими, ведь мы можем их в любой момент поправить
- Функциональности побольше, чтобы достичь максимальной консистентности поведения на уровне проекта



Локальные компоненты

- Могут быть максимально не гибкими, ведь мы можем их в любой момент поправить
- Функциональности побольше, чтобы достичь максимальной консистентности поведения на уровне проекта
- API может быть совсем не явным, завязанным на конкретную бизнес логику

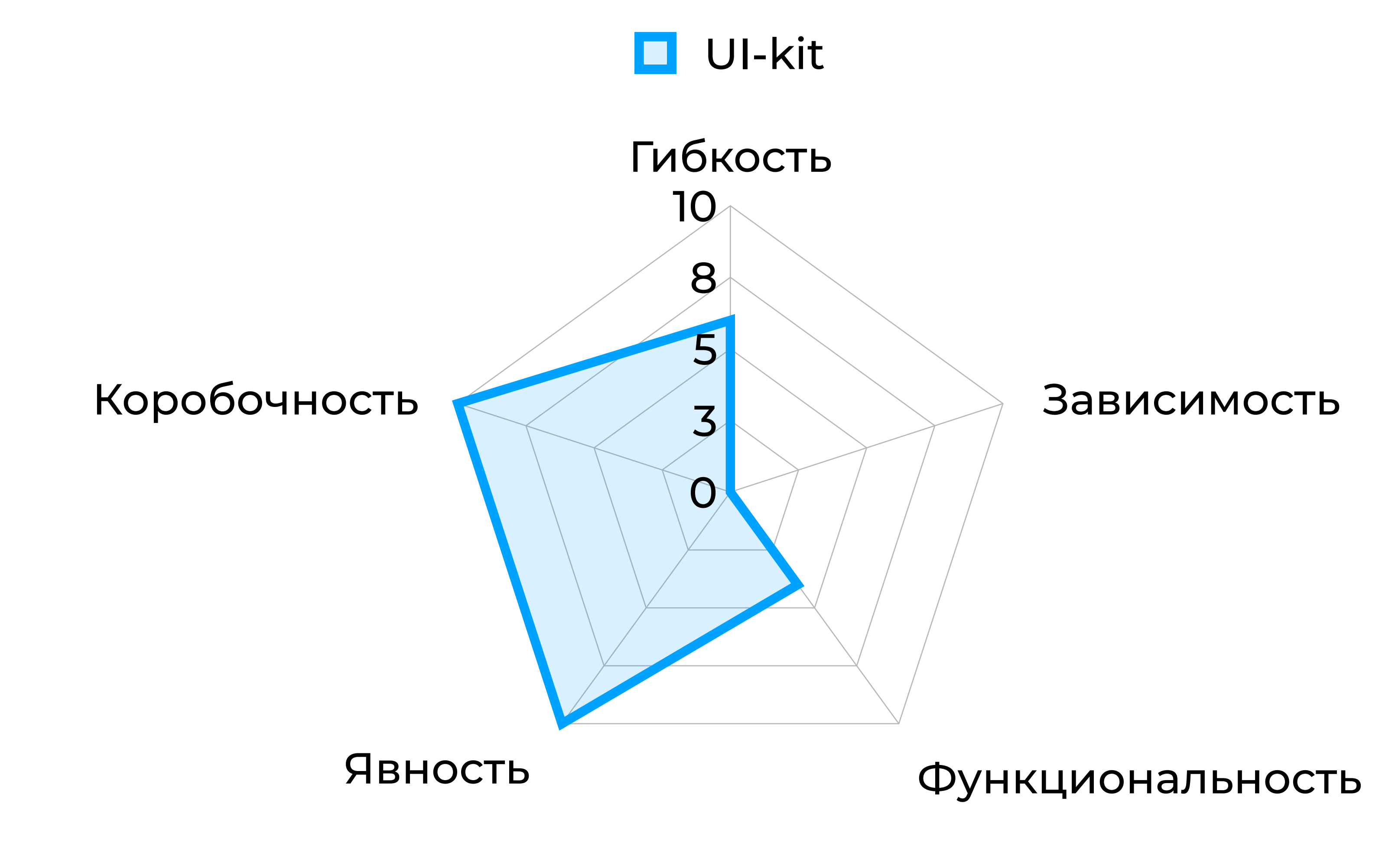



А что делать?
Как мы можем влиять на характеристики?
avito.tech
59

Пишите тесты

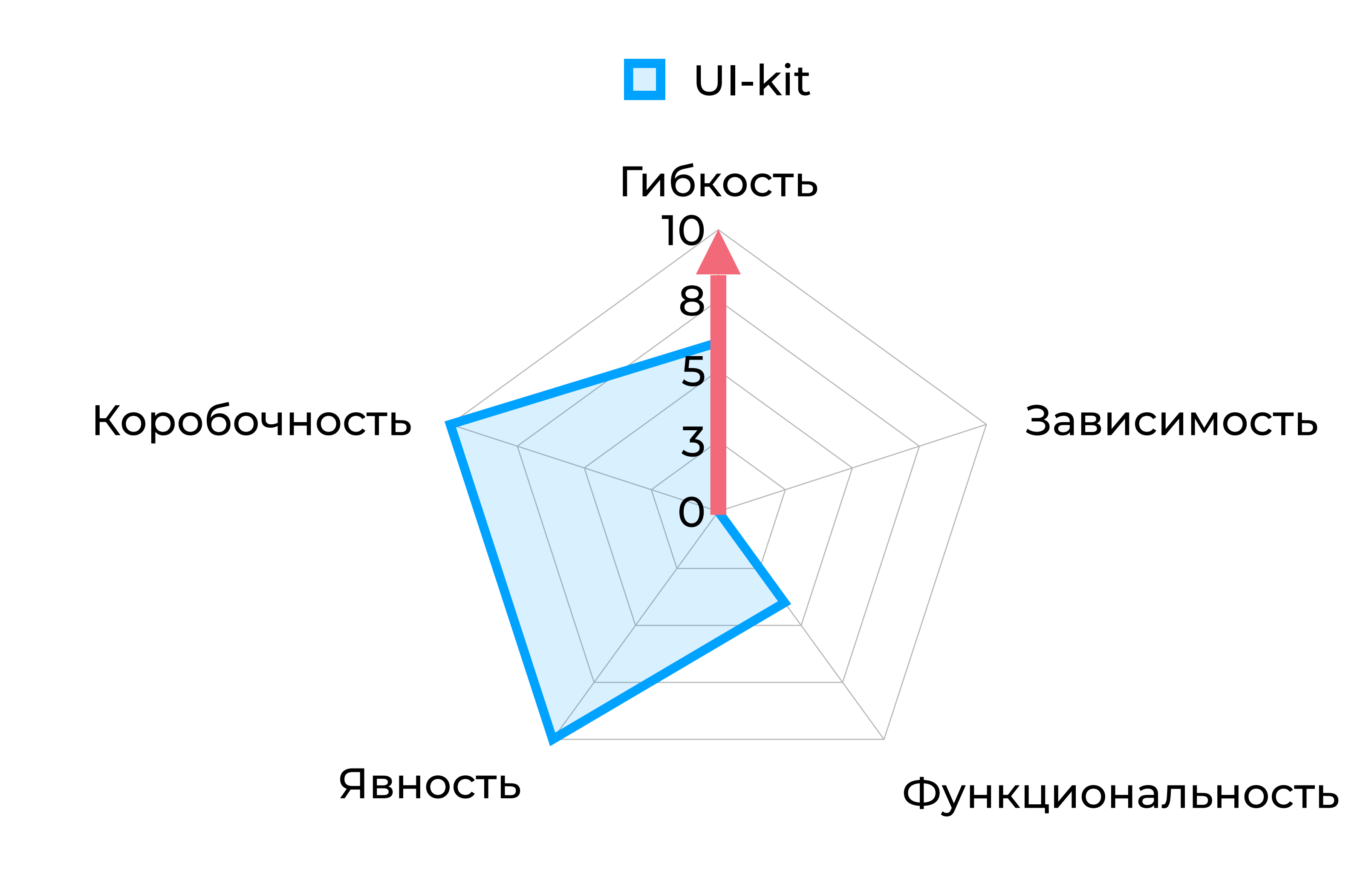


Dependency injection



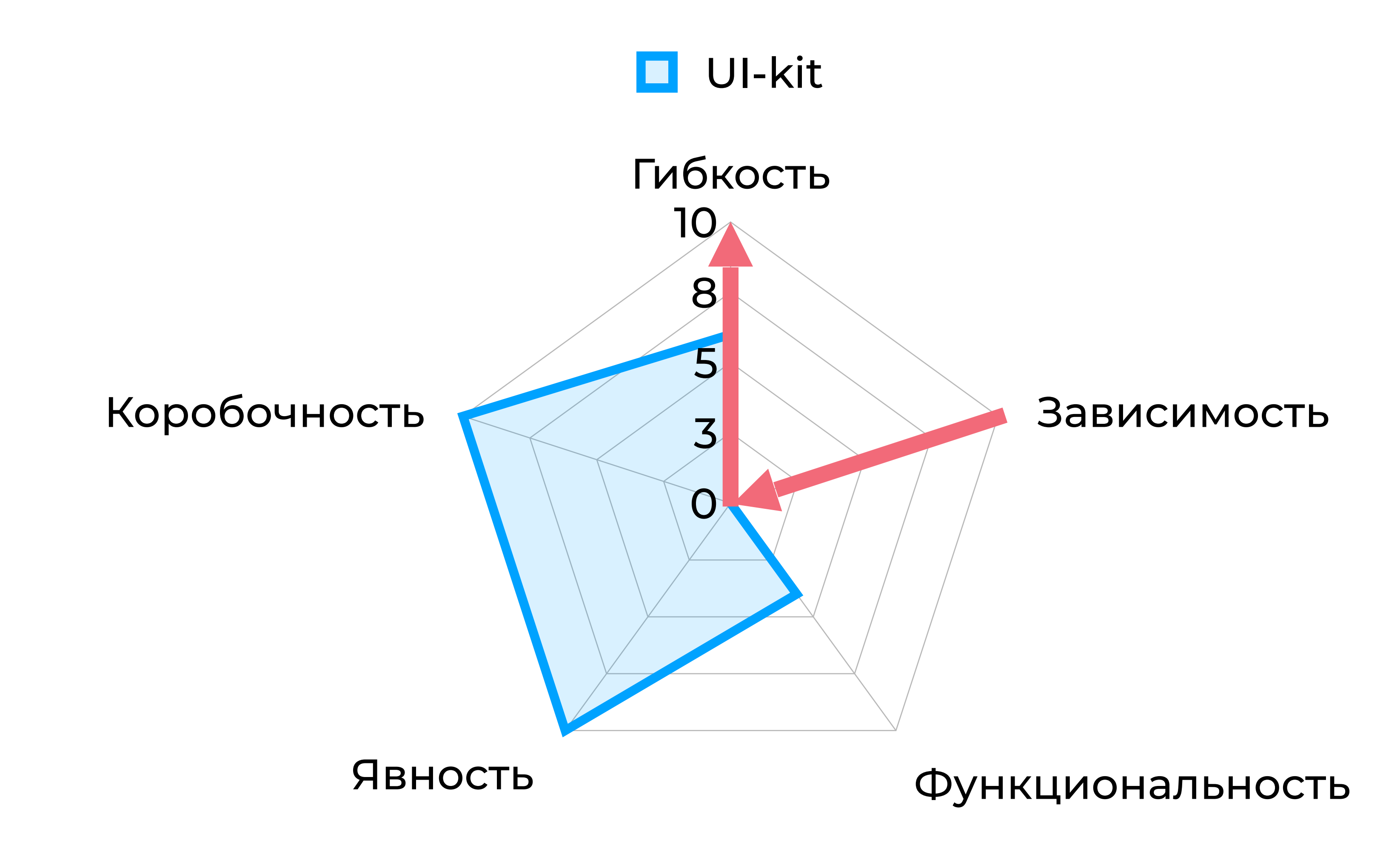
Dependency injection

- Делаем гибче



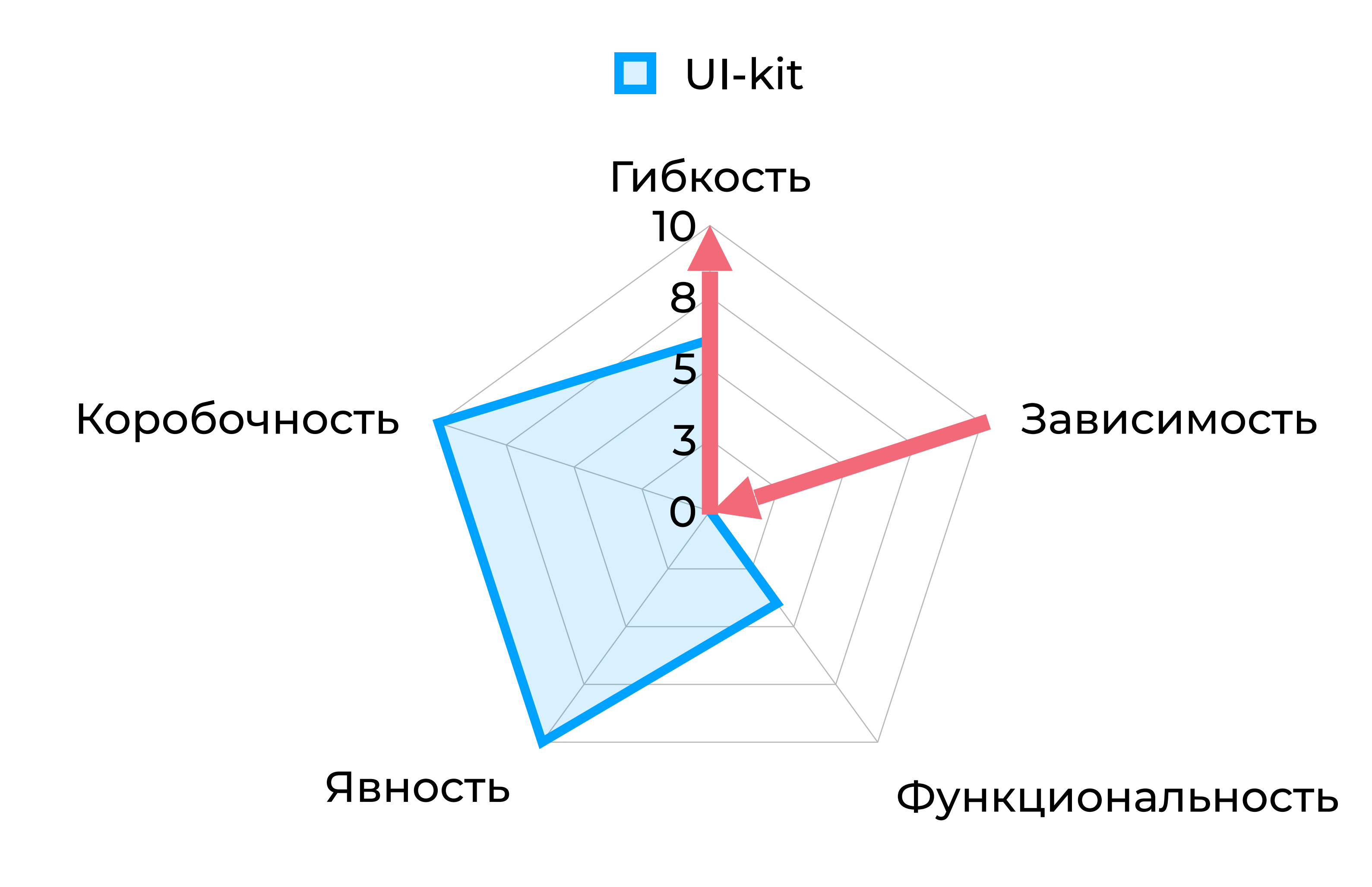
Dependency injection

- Делаем гибче
- Делаем независимее



Dependency injection

- Делаем гибче
- Делаем независимее
- С использованием DI легче тестировать (меньше мокать)



Injection via props

```
function TextField({
    withoutImplicitFocus,
    disabled,
    onFocus,
    hasLowerCase,
    hasAutoSelectAfterSubmit,
    onChange: onChangeProp,
    hasAutoSelect = true,
    selector = DEFAULT_SELECTOR,
    inputSize = "l",
    priority = 0,
    dataE2e = selector || DEFAULT_SELECTOR,
    dataTestId = selector || DEFAULT_SELECTOR,
    handleEnter = selectOnEnter,
    transformValueOnChange = transformToUppercase,
    onKeyDown = noop,
    useSuperFocus = useSuperFocusDefault,
    useFocusAfterError = useFocusAfterErrorDefault,
    useSuperFocusOnKeydown = useSuperFocusOnKeydownDefault,
    useSuperFocusAfterDisabled = useSuperFocusAfterDisabledDefault,
    someJSX,
    ...textFieldProps
}: Props)
```

Injection via props

- Функции

```
function TextField({
    withoutImplicitFocus,
    disabled,
    onFocus,
    hasLowerCase,
    hasAutoSelectAfterSubmit,
    onChange: onChangeProp,
    hasAutoSelect = true,
    selector = DEFAULT_SELECTOR,
    inputSize = "l",
    priority = 0,
    dataE2e = selector || DEFAULT_SELECTOR,
    dataTestId = selector || DEFAULT_SELECTOR,
    handleEnter = selectOnEnter,
    transformValueOnChange = transformToUppercase,
    onKeyDown = noop,
    useSuperFocus = useSuperFocusDefault,
    useFocusAfterError = useFocusAfterErrorDefault,
    useSuperFocusOnKeydown = useSuperFocusOnKeydownDefault,
    useSuperFocusAfterDisabled = useSuperFocusAfterDisabledDefault,
    someJSX,
    ...textFieldProps
}: Props)
```

Injection via props

- Функции
- Хуки

```
function TextField({
    withoutImplicitFocus,
    disabled,
    onFocus,
    hasLowerCase,
    hasAutoSelectAfterSubmit,
    onChange: onChangeProp,
    hasAutoSelect = true,
    selector = DEFAULT_SELECTOR,
    inputSize = "l",
    priority = 0,
    dataE2e = selector || DEFAULT_SELECTOR,
    dataTestId = selector || DEFAULT_SELECTOR,
    handleEnter = selectOnEnter,
    transformValueOnChange = transformToUppercase,
    onKeyDown = noop,
    useSuperFocus = useSuperFocusDefault,
    useFocusAfterError = useFocusAfterErrorDefault,
    useSuperFocusOnKeydown = useSuperFocusOnKeydownDefault,
    useSuperFocusAfterDisabled = useSuperFocusAfterDisabledDefault,
    someJSX,
    ...textFieldProps
}: Props)
```

Injection via props

- Функции
- Хуки
- Объекты

```
function TextField({
    withoutImplicitFocus,
    disabled,
    onFocus,
    hasLowerCase,
    hasAutoSelectAfterSubmit,
    onChange: onChangeProp,
    hasAutoSelect = true,
    selector = DEFAULT_SELECTOR,
    inputSize = "l",
    priority = 0,
    testProps: {
        dataE2e = selector || DEFAULT_SELECTOR,
        dataTestId = selector || DEFAULT_SELECTOR,
    },
    handleEnter = selectOnEnter,
    transformValueOnChange = transformToUppercase,
    onKeyDown = noop,
    someJSX,
    ...textFieldProps
}: Props) {}
```

Injection via context

- Функции
- Хуки
- Объекты

```
import React from "react";
import { useForm, FormProvider, useFormContext } from "react-
hook-form";

export default function App() {
  const methods = useForm();
  const onSubmit = data => console.log(data);

  return (
    <FormProvider {...methods} >
      <form onSubmit={methods.handleSubmit(onSubmit)}>
        <NestedInput />
        <input type="submit" />
      </form>
    </FormProvider>
  );
}

function NestedInput() {
  const { register } = useFormContext(); // retrieve all hook
methods
  return <input {...register("test")} />;
}
```

Injection via context

- Функции
- Хуки
- Объекты
- Autowiring

```
import { provider, inject } from "react-ioc"

class DataContext {
  users = observable.map<number, User>();
  posts = observable.map<number, Post>();
}

class PostService {
    @inject dataContext: DataContext;

    @action
    createPost(user: User) {
      const post = new Post({ id: uniqueId() });
      this.dataContext.posts.set(post.id, post);
      return post;
    }
}

@provider(DataContext, PostService)
class App extends React.Component {
  render() {
    // ...
  }
}
```

Injection via context

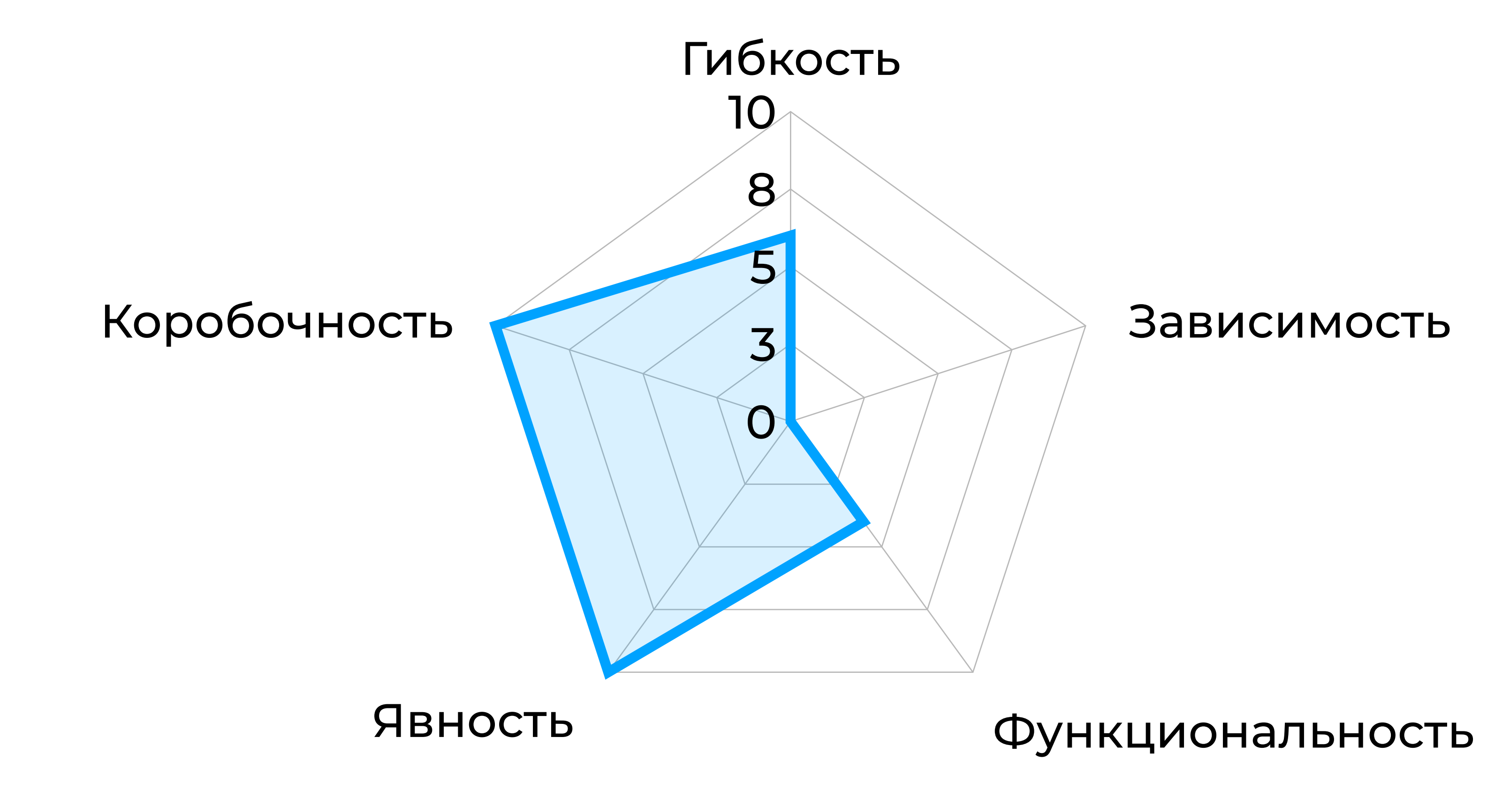
- Функции
- Хуки
- Объекты
- Autowiring

```
import { useInstance } from "react-ioc";

const MyButton = props => {
  const postService = useInstance(PostService);

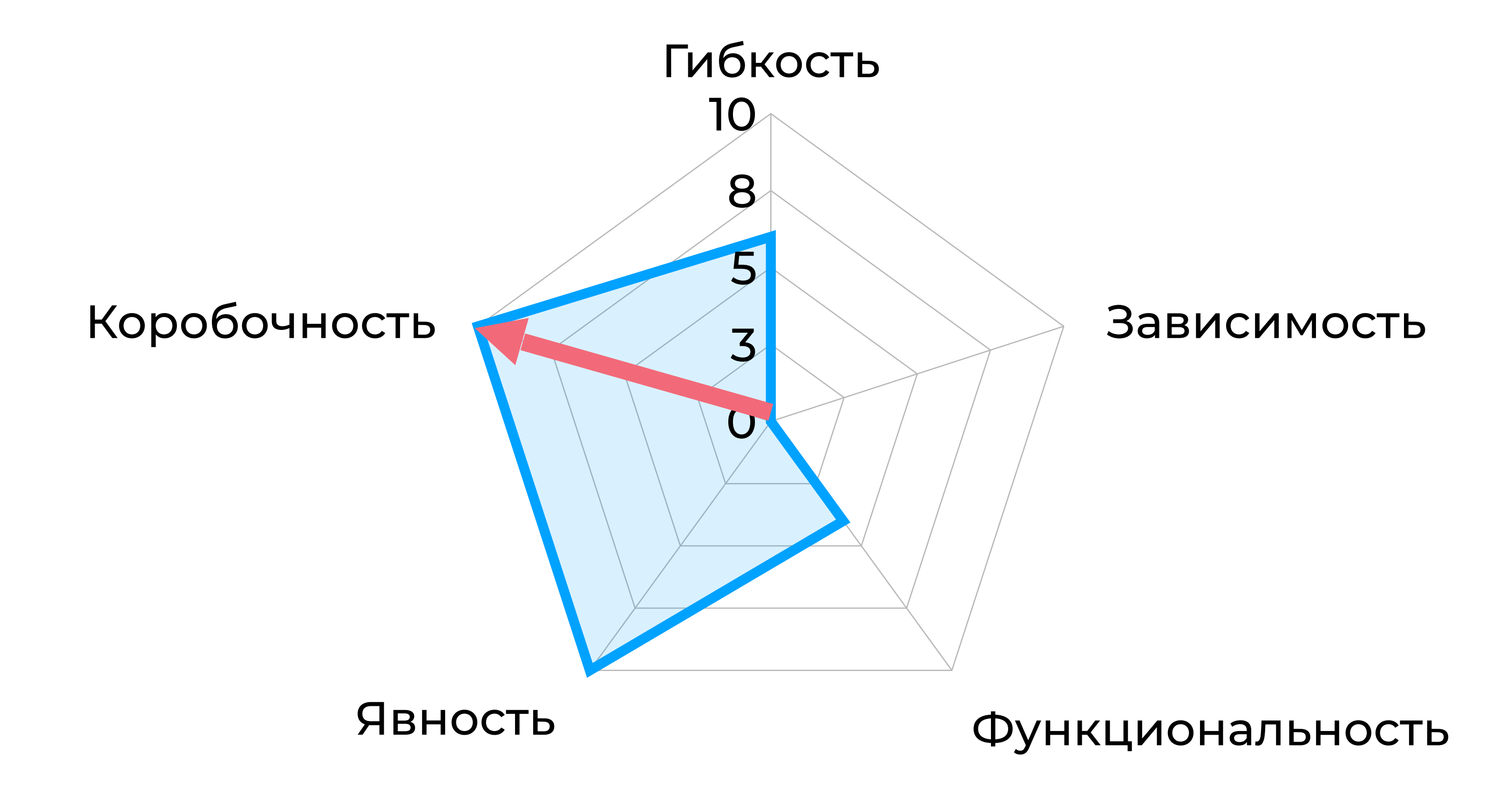
  return <button onClick={() => postService.createPost({})}>Ok</button>
}
```

DSL like подход



DSL like подход

- Делаем коробочнее



DSL like подход

- Строго ограничиваем возможности конфигом

```
function BasicForm() {
  ...
  const form = useForm({
    onSubmit(values) {
      alert(JSON.stringify(values, null, 2))
      console.log('values', values)
    },
    children: [
      {
        label: 'First Name',
        name: 'firstName',
        component: 'Input',
        value: '',
      },
      {
        component: 'Submit',
        text: 'submit',
      },
    ],
  })

  return <Form form={form} />
}
```

DSL like подход

- Строго ограничиваем возможности конфигом
- Но даем возможность заинжектить нужные нам компоненты

```
function BasicForm() {
  ...
  const form = useForm({
    ...,
    components: [
      Input,
      Submit
    ]
    ...
  })

  return <Form form={form} />
}
```

DSL like подход

- Или оставляем общим только поведение

```
import { useForm } from "react-hook-form";

export default function App() {
  const { register, handleSubmit } =
useForm({ shouldUseNativeValidation: true });
  const onSubmit = async data => { console.log(data); };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input
        {...register("firstName", { required: "Please enter your first
name." })} // custom message
      />
      <input type="submit" />
    </form>
  );
}
```

DSL like подход

- Или оставляем общим только поведение
- Например мы хотим сделать 1000 формочек на сайте, и не хотим каждый раз их переписывать

```
import { useForm } from "react-hook-form";

export default function App() {
  const { register, handleSubmit } =
useForm({ shouldUseNativeValidation: true });
  const onSubmit = async data => { console.log(data); };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input
        {...register("firstName", { required: "Please enter your first
name." })} // custom message
      />
      <input type="submit" />
    </form>
  );
}
```


ДЕЛАЙТЕ ХОРОШО
А ПЛОХО НЕ ДЕЛАЙТЕ

avito.tech

Москва — 2022

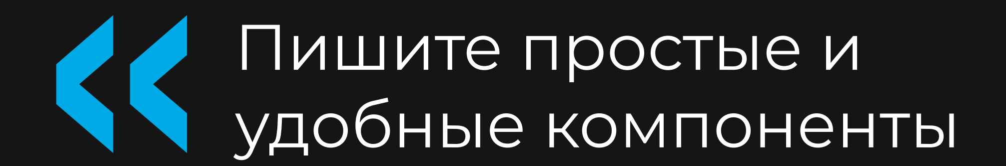
Антон Крылов

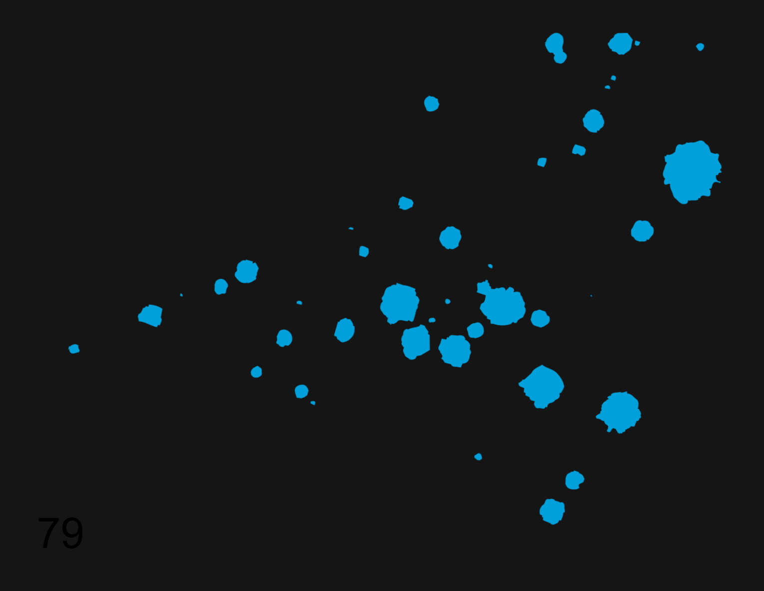
Frontend developer

 anton.krylov322@gmail.com

 https://github.com/pivaszbs/

 @dOrDie





avito.tech

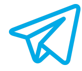
Москва — 2022

Антон Крылов

Frontend developer


anton.krylov322@gmail.com


https://github.com/pivaszbs/


@dOrDie

Голосуйте за доклад





Ссылка на материалы из доклада